数量经济学系列丛书

Economic and Financial Data Analysis
and Its Application in Python

经济金融数据分析及其Python应用

朱顺泉　编著

QUANTITATIV



ECONOMICS

清华大学出版社

数量经济学系列丛书

经济金融数据分析及其 Python 应用

朱顺泉　编著

清华大学出版社
北　京

内容简介

本书结合实例对Python进行全面介绍，并侧重介绍使用Python进行经济金融数据分析。全书共13章，包括：经济金融数据分析及Python环境、Python数据分析程序包应用基础、Python数据分析的数据存取、Python图形的绘制和可视化、概率统计分布的Python应用、描述性统计的Python应用、参数估计的Python应用、参数假设检验的Python应用、相关分析与一元回归数据分析的Python应用、多元回归数据分析的Python应用、机器学习数据分析的Python应用、时间序列数据分析的Python应用、量化金融数据分析的Python应用。

本书适应大数据分析时代，内容新颖、全面，实用性强，可作为统计学、金融学、投资学、数量经济学、工商管理、管理科学与工程等相关专业的本科高年级学生与研究生学习统计学或数据分析等课程的教材或参考书。

图书在版编目(CIP)数据

经济金融数据分析及其Python应用/朱顺泉编著.—北京：清华大学出版社，2018
(数量经济学系列丛书)
ISBN 978-7-302-49743-1

Ⅰ.①经…　Ⅱ.①朱…　Ⅲ.①金融—数据处理软件　Ⅳ.①F830.49

中国版本图书馆CIP数据核字(2018)第035805号

责任编辑：高晓蔚
封面设计：常雪影
责任校对：宋玉莲
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京鑫海金澳胶印有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　**印　张**：14.5　**插　页**：1　**字　　数**：346千字
版　　次：2018年11月第1版　　**印　　次**：2018年11月第1次印刷
定　　价：45.00元

产品编号：073629-01



大数据时代，数据已成为人们进行商务决策时最重要的参考依据之一，数据分析行业迈入了一个全新的阶段。《经济金融数据分析及其 Python 应用》重点介绍了 Python 的数据存取、数据的可视化、数据统计分析、机器学习、时间序列分析和金融量化分析的 Python 应用，同时结合大量的实例，对 Python 的重要程序包进行科学、准确和全面的介绍，以便使读者深刻理解 Python 的精髓和灵活、高效的使用技巧。

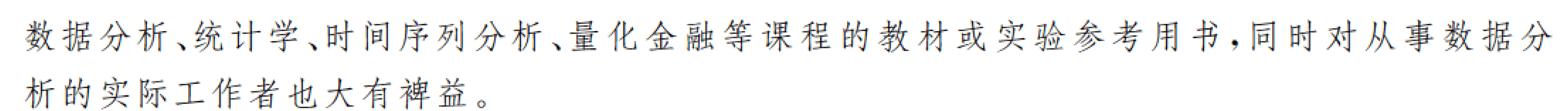
本书之所以采用 Python 软件，是因为它具有强大的图形展示、统计分析、机器学习功能，免费使用及功能强大的 Pandas（基本数据分析工具）、NumPy（数值计算工具）、SciPy（科学计算工具）、Matplotlib（基础绘图工具）、Seaborn（扩展绘图工具）、Sklearn（机器学习工具）等众多程序包（而 Matlab、SAS、SPSS、EViews、Stata、S-PLUS 等都是付费软件），因此它越来越受到广大用户的欢迎和喜爱。

本书通过丰富的实例，详细介绍了 Python 在数据存取、图形展示、统计分析、机器学习、时间序列、量化金融等领域中的应用，侧重于理论方法与应用相结合，实例丰富且通俗易懂，尤其对 Python 软件的各种绘图方法、不同数据表的接口、统计分析、机器学习、时间序列、量化金融等方面的介绍有较好的特色，详细地介绍了各种绘图方法、不同数据的接口、统计分析、机器学习、时间序列、量化金融等方面在 Python 中的实现过程。本书的特点是：以问题为导向，通过问题来介绍 Python 的使用方法。因此，读者通过本书不仅能掌握 Python 及相关的程序包的使用方法，而且能学会从实际问题分析入手，应用 Python 解决经济金融领域中的各种数据分析问题。

本书的内容是这样安排的：第 1 章介绍经济金融数据分析及 Python 环境，第 2 章介绍 Python 数据分析程序包应用基础，第 3 章介绍 Python 数据分析的数据存取，第 4 章介绍 Python 图形的绘制和可视化，第 5 章介绍概率统计分布的 Python 应用，第 6 章介绍描述性统计的 Python 应用，第 7 章介绍参数估计的 Python 应用，第 8 章介绍参数假设检验的 Python 应用，第 9 章介绍相关分析与一元回归数据分析的 Python 应用，第 10 章介绍多元回归数据分析的 Python 应用，第 11 章介绍机器学习数据分析的 Python 应用，第 12 章介绍时间序列数据分析的 Python 应用，第 13 章介绍量化金融数据分析的 Python 应用。

本书实例和内容丰富，针对性强，书中各章详细地介绍了实例的 Python 具体操作过程，读者只需按照书中介绍的步骤一步一步地实际操作，就能掌握全书的内容。为了帮助读者更加直观地学习本书，我们将书中实例的全部数据文件打包收录，读者可扫描书末页的二维码获取。读者在自己的电脑中建立一个 data 目录（其他目录名也可以），将所有数据文件复制到此目录，即可进行操作。

本书适合作为统计学、金融学、经济学、管理学等相关专业的本科生或研究生学习



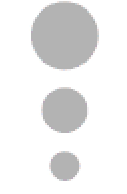
数据分析、统计学、时间序列分析、量化金融等课程的教材或实验参考用书，同时对从事数据分析的实际工作者也大有裨益。

本书部分内容为广东省自然科学基金项目成果，也是广东财经大学数据模型与决策示范课程的阶段性成果。

本书的出版得到了清华大学出版社编校人员的大力支持和帮助，感谢他们为本书编辑校对付出的辛苦工作。由于时间和水平的限制，书中难免出现一些纰漏，恳请读者谅解并提出宝贵意见。

作　者
2018年5月于广州


目
CONTENTS
录





经济金融数据分析及 Python 环境

数据分析是指用适当的统计与计量分析方法对收集来的大量数据进行分析，提取有用信息和形成结论而对数据加以详细研究和概括总结的过程。这一过程也是质量管理体系的支持过程。在实用中，数据分析可帮助人们作出判断，以便采取适当行动。

大数据分析是指对规模巨大的数据进行分析。大数据的特点可以概括为 5 个 V，数据量大（volume）、速度快（velocity）、类型多（variety）、有价值（value）、真实性（veracity）。大数据已成为 IT 行业时下最火热的词汇。随之而来的数据仓库、数据安全、数据分析、数据挖掘等等围绕大数据的商业价值的利用逐渐成为行业人士争相追捧的利润焦点。随着大数据时代的来临，大数据分析也应运而生。

本章简要介绍经济金融数据的类型、来源，主要的数据分析软件包，以及目前流行的经济金融数据分析 Python 语言及其环境。

1.1　经济金融数据类型

经济金融中需要处理的数据类型主要有三类：横截面数据、时间序列数据和面板数据。

1.1.1　横截面数据

横截面数据是同一时间（时期或时点）某一指标在不同空间的观测数据。如某一时点中国 A 股市场的平均收益率，2017 年所有 A 股上市公司的净资产收益率。在利用横截面数据作分析时，由于单个或多个解释变量观测值起伏变化会对被解释产生不同的影响，因而导致异方差问题。因此在数据整理时必须消除异方差。

1.1.2　时间序列数据

时间序列数据即按时间序列排列的数据，也称为动态序列数据。时间序列数据是按照一定时间间隔对某一变量或不同时间的取值进行观测所得到的一组数据，例如每一季度的 GDP 数据、每一天的股票交易数据或债券收益率数据等。在经济金融数据分析中，时间序列数据是常见的一类数据类型。

1.1.3　面板数据

面板数据即时间序列数据和横截面数据相结合的数据。

金融领域以时间序列数据分析（如金融市场）与面板数据分析（如公司金融）为主。

1.2　经济金融数据来源

1.2.1　专业性网站

如国家统计局网站、中国人民银行网站、中国证监会网站、世界银行网站、国际货币基金组织网站等。

1.2.2　专业数据公司和信息公司

国外数据库主要有芝加哥大学商学院的证券价格研究中心(CRSP)、路透(Reuters)终端、彭博(Bloomberg)终端、雅虎财经等。国内提供经济金融数据库主要有：CCER 中国经济金融数据库、国泰安数据库(GTA)、万德数据库(Wind)、锐思数据库等。如表 1-1 所示。

表 1-1　经济金融数据库

数据来源名称	网　　址
CRSP	www.chicagobooth.edu
路透	www.reuters.com
彭博	www.bloomberg.com
雅虎财经	www.finance.yahoo.com
万得 Wind 经济金融数据库	www.wind.com.cn
国泰安 GTA 经济金融数据库	www.gtadata.com
CCER 中国经济金融数据库	www.ccer.edu.cn
聚源锐思经济金融数据库	www.resset.cn
天相经济金融数据库	www.txsec.com/zqsc/tx_data.asp

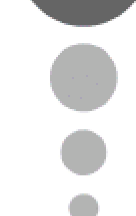
1.2.3　抽样调查

抽样调查是针对某些专门的研究开展的一类获取数据的方式。比如，要对中国的投资者信心进行建模，就必须通过设计调查问卷，对不同的投资群体进行数据采集。

1.3　经济金融数据分析工具简介

1.3.1　Python 数据分析工具简介

Python 是一种面向对象的解释型计算机程序设计语言，由 Guido van Rossum 于 1989 年底发明，第一个公开发行版发行于 1991 年，Python 源代码同样遵循 GPL(GNU General Public License)协议。Python 语法简洁而清晰，具有丰富和强大的类库。它常被昵称为胶水语言，能够把用其他语言制作的各种模块(尤其是 C/C++)很轻松地联结在一起。常见的一种应用情形是，使用 Python 快速生成程序的原型(有时甚至是程序的最终界面)，然后对其中有特别要求的部分，用更合适的语言改写，比如 3D 游戏中的图形渲染模块，性能要求特别高，就可以用 C/C++重写，而后封装为 Python 可以调用的扩展类库。需要注意的是在



您使用扩展类库时可能需要考虑平台问题，某些可能不提供跨平台的实现。

Python 需要安装 Pandas、NumPy、SciPy、Statsmodels、Matplotlib 等一系列的程序包，还需要安装 IPython 交互环境，目前有包括这些程序包的套装软件可供下载。

详细内容请登录 https://www.Python.org 查询。

1.3.2　R 数据分析工具简介

R 是统计领域广泛使用的诞生于 1980 年左右的 S 语言的一个分支。可以认为 R 是 S 语言的一种实现。而 S 语言是由 AT&T 贝尔实验室开发的一种用来进行数据探索、统计分析和作图的解释型语言。最初 S 语言的实现版本主要是 S-PLUS。S-PLUS 是一个商业软件，它基于 S 语言，并由 MathSoft 公司的统计科学部进一步完善。后来 Auckland 大学的 Robert Gentleman 和 Ross Ihaka 及其他志愿人员开发了一个 R 系统。由"R 开发核心团队"负责开发。R 是基于 S 语言的一个 GNU 项目，所以也可以当作 S 语言的一种实现，通常用 S 语言编写的代码都可以不作修改地在 R 环境下运行。R 的语法是来自 Scheme。R 的使用与 S-PLUS 有很多类似之处，这两种语言有一定的兼容性。S-PLUS 的使用手册，只要稍加修改就可作为 R 的使用手册。所以有人说：R 是 S-PLUS 的一个"克隆"。

详细内容请登录：http://cran.r-project.org 查询。

1.3.3　Stata 数据分析工具简介

Stata 由美国计算机资源中心(Computer Resource Center)1985 年研制。其特点是采用命令行/程序操作方式，程序短小精悍，功能强大。Stata 是一套提供其使用者数据分析、数据管理以及绘制专业图表的完整及整合性统计软件。它提供许许多多功能，包含线性混合模型、均衡重复反复及多项式普罗比模式。新版本的 Stata 采用最具亲和力的窗口接口，使用者自行建立程序时，软件能提供具有直接命令式的语法。Stata 提供完整的使用手册，包含统计样本建立、解释、模型与语法、文献等出版品。

除此之外，Stata 工具可以通过网络实时更新最新功能，更可以得知世界各地的使用者对于 Stata 公司提出的问题与解决之道。使用者也可以通过 Stata Journal 获得许许多多的相关信息以及书籍介绍等。另外一个获取庞大资源的渠道就是 Statalist，它是一个独立的 listserver，每月交替提供使用者超过 1000 条信息以及 50 个程序。

目前最新版为 Stata 14.0 版。

详细内容请登录 http://www.stata.com 查询。

1.3.4　Matlab 数据分析工具简介

Matlab 工具是由美国 Mathworks 公司推出的用于数值计算和图形处理的科学计算系统，在 Matlab 工具环境下，用户可以集成地进行程序设计、数值计算、图形绘制、输入输出、文件管理等各项操作。它提供的是一个人机交互的数学系统环境，与利用 C 语言作数值计算的程序设计相比，利用 Matlab 可以节省大量的编程时间，且程序设计自由度大。它最大的特点是给用户带来最直观、最简洁的程序开发环境，语言简洁紧凑，使用方便灵活，库函数与运算符极其丰富，另外具有强大的图形功能。

在国际学术界，Matlab 已经被确认为准确、可靠的科学计算标准软件，许多国际一流学

术刊物上，都可以看到 Matlab 的应用。

详细内容请登录 http://www.mathworks.com 查询。

1.3.5 EViews 数据分析工具简介

EViews 是美国 GMS 公司 1981 年发行的第 1 版 Micro TSP 的 Windows 版本，通常称为计量经济学软件包。EViews 是 Econometrics Views 的缩写，它的本意是对社会经济关系与经济活动的数量规律，采用计量经济学方法与技术进行"观察"。计量经济学研究的核心是设计模型、收集资料、估计模型、检验模型、运用模型进行预测、求解模型和应用模型。EViews 是完成上述任务必不可少的得力工具。正是由于 EViews 等计量经济学软件包的出现，使计量经济学取得了长足的进步，发展为实用与严谨的经济学科。使用 EViews 软件包可以对时间序列和非时间序列的数据进行分析，建立序列（变量）间的统计关系式，并用该关系式进行预测、模拟等。虽然 EViews 是由经济学家开发的，并且大多数被用于经济学领域，但并非意味着该软件包仅可用于处理经济方面的时间序列。EViews 处理非时间序列数据照样得心应手。实际上，相当大型的非时间序列（截面数据）的项目也能在 EViews 中进行处理。

详细内容请登录 http://www.eviews.com 查询。

1.3.6 SAS 数据分析工具简介

SAS 是美国 SAS 研究所研制的一套大型集成应用软件系统，具有完备的数据存取、数据管理、数据分析和数据展现功能。尤其是创业产品统计分析系统部分，由于其具有强大的数据分析能力，一直为业界著名软件，在数据处理和统计分析领域，被誉为国际上的标准软件和最权威的优秀统计软件包，广泛应用于行政管理、科研、教育、生产和金融等不同领域，发挥着重要的作用。SAS 系统中提供的主要分析功能包括统计分析、经济计量分析、时间序列分析、决策分析、财务分析和全面质量管理工具等。

详细内容请登录 http://www.sas.com 查询。

1.3.7 SPSS 数据分析工具简介

SPSS（statistical package for the social science）——社会科学统计软件包，是世界著名的统计分析软件之一。20 世纪 60 年代末，美国斯坦福大学的三位研究生研制开发了最早的统计分析软件 SPSS，同时成立了 SPSS 公司，并于 1975 年在芝加哥组建了 SPSS 总部。20 世纪 80 年代以前，SPSS 统计软件主要应用于企事业单位。1984 年 SPSS 总部首先推出了世界第一个统计分析软件微机版本 SPSS/PC+，开创了 SPSS 微机系列产品的开发方向，从而确立了个人用户市场第一的地位。2009 年 IBM 收购 SPSS 公司后，现在在中国市场上推出的最新产品，是 IBM SPSS Statistics 21.0 多国语言版。SPSS/PC+的推出，极大地扩充了它的应用范围，使其能很快地应用于自然科学、技术科学、社会科学的各个领域，世界上许多有影响的报纸杂志纷纷就 SPSS 的自动统计绘图、数据的深入分析、使用方便、功能齐全等方面给予了高度的评价与称赞。目前已经在国内逐渐流行起来。它使用 Windows 的窗口方式展示各种管理和分析数据方法的功能，使用对话框展示出各种功能选择项，只要掌握一定的 Windows 操作技能，粗通统计分析原理，就可以使用该软件为特定的科研工作

服务。

详细内容请登录 http://www.spss.com 查询。

还有一些统计和计量经济学软件，如 Statistica、S-PLUS 等，但相对来说没有以上 7 种软件流行。各软件网站列表如表 1-2 所示。

表 1-2　常见的经济金融数据分析工具网站

工具名称	网　址
Python	www.Python.org
R	www.cran.r-project.org
Stata	www.stata.com
Matlab	www.mathworks.com
EViews	www.eviews.com
SAS	www.sas.com
SPSS	www.spss.com

1.4　Python 数据分析工具的下载

输入如下网址：https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/，即可下载 Anaconda，它是一个用于科学计算 Python 发行版的套装软件，支持 Linux、Mac、Windows 等操作系统，包含了众多流行的科学计算、数据分析的 Python 包。其中包括 Pandas、NumPy、SciPy、Statsmodels、Matplotlib 等一系列的程序包以及 iPython 交互环境。界面如图 1-1 所示。

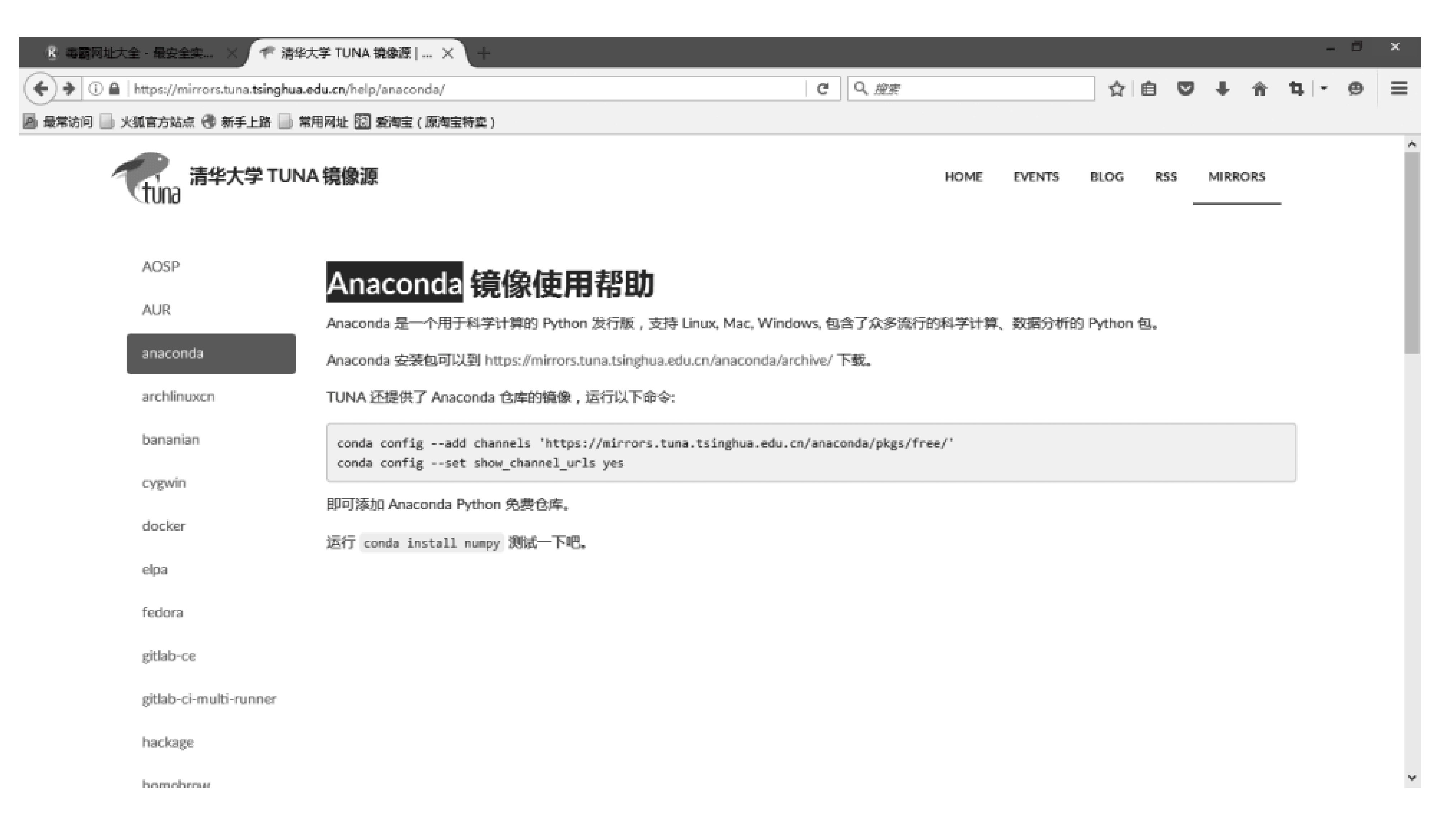
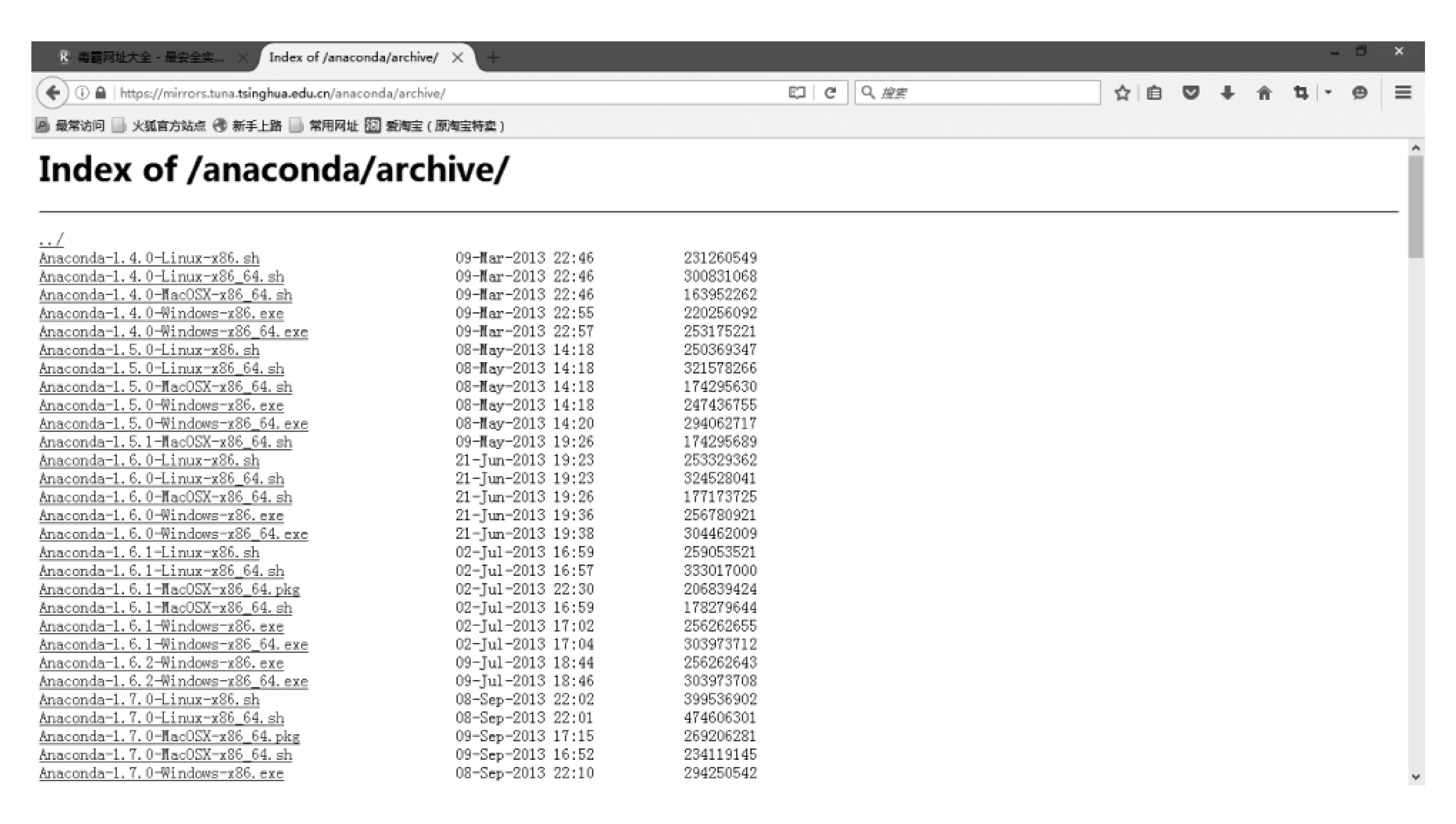


图 1-1　Anaconda 安装包界面

点击图 1-1 中的 https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/，出现图 1-2 所示的界面。



../		
Anaconda-1.4.0-Linux-x86.sh	09-Mar-2013 22:46	231260549
Anaconda-1.4.0-Linux-x86_64.sh	09-Mar-2013 22:46	300831068
Anaconda-1.4.0-MacOSX-x86_64.sh	09-Mar-2013 22:46	163952262
Anaconda-1.4.0-Windows-x86.exe	09-Mar-2013 22:55	220256092
Anaconda-1.4.0-Windows-x86_64.exe	09-Mar-2013 22:57	253175221
Anaconda-1.5.0-Linux-x86.sh	08-May-2013 14:18	250369347
Anaconda-1.5.0-Linux-x86_64.sh	08-May-2013 14:18	321578266
Anaconda-1.5.0-MacOSX-x86_64.sh	08-May-2013 14:18	174295630
Anaconda-1.5.0-Windows-x86.exe	08-May-2013 14:18	247436755
Anaconda-1.5.0-Windows-x86_64.exe	08-May-2013 14:20	294062717
Anaconda-1.5.1-MacOSX-x86_64.sh	09-May-2013 19:26	174295689
Anaconda-1.6.0-Linux-x86.sh	21-Jun-2013 19:23	253329362
Anaconda-1.6.0-Linux-x86_64.sh	21-Jun-2013 19:23	324528041
Anaconda-1.6.0-MacOSX-x86_64.sh	21-Jun-2013 19:26	177173725
Anaconda-1.6.0-Windows-x86.exe	21-Jun-2013 19:36	256780921
Anaconda-1.6.0-Windows-x86_64.exe	21-Jun-2013 19:38	304462009
Anaconda-1.6.1-Linux-x86.sh	02-Jul-2013 16:59	259053521
Anaconda-1.6.1-Linux-x86_64.sh	02-Jul-2013 16:57	333017000
Anaconda-1.6.1-MacOSX-x86_64.pkg	02-Jul-2013 22:30	206839424
Anaconda-1.6.1-MacOSX-x86_64.sh	02-Jul-2013 16:59	178279644
Anaconda-1.6.1-Windows-x86.exe	02-Jul-2013 17:02	256262655
Anaconda-1.6.1-Windows-x86_64.exe	02-Jul-2013 17:04	303973712
Anaconda-1.6.2-Windows-x86.exe	09-Jul-2013 18:44	256262643
Anaconda-1.6.2-Windows-x86_64.exe	09-Jul-2013 18:46	303973708
Anaconda-1.7.0-Linux-x86.sh	08-Sep-2013 22:02	399536902
Anaconda-1.7.0-Linux-x86_64.sh	08-Sep-2013 22:01	474606301
Anaconda-1.7.0-MacOSX-x86_64.pkg	09-Sep-2013 17:15	269206281
Anaconda-1.7.0-MacOSX-x86_64.sh	09-Sep-2013 16:52	234119145
Anaconda-1.7.0-Windows-x86.exe	08-Sep-2013 22:10	294250542

图 1-2　Anaconda 安装包界面

在图 1-2 中，选择 Anaconda2-2.4.1-Windows-x86.exe，即可得到用 Python 作经济金融数据分析的套装软件工具。也可以选择最新的 Anaconda3-4.1.1-Windows-x86.exe（32 位），Anaconda3-4.1.1-Windows-x86_64.exe（64 位）。请读者注意，本书的经济金融数据分析以下载的 Anaconda2-2.4.1-Windows-x86.exe（32 位）工具来说明其应用。下载界面如图 1-3 所示（该图的倒数第二行即为选择的下载对象）。

Anaconda2-2.4.0-MacOSX-x86_64.pkg	02-Nov-2015 22:22	287613909
Anaconda2-2.4.0-MacOSX-x86_64.sh	02-Nov-2015 22:22	251172115
Anaconda2-2.4.0-Windows-x86.exe	02-Nov-2015 22:22	337056800
Anaconda2-2.4.0-Windows-x86_64.exe	02-Nov-2015 22:22	406819096
Anaconda2-2.4.1-Linux-x86.sh	08-Dec-2015 21:00	260583576
Anaconda2-2.4.1-Linux-x86_64.sh	08-Dec-2015 21:00	277827702
Anaconda2-2.4.1-MacOSX-x86_64.pkg	08-Dec-2015 21:00	257787337
Anaconda2-2.4.1-MacOSX-x86_64.sh	08-Dec-2015 21:00	222326344
Anaconda2-2.4.1-Windows-x86.exe	08-Dec-2015 21:00	301790720
Anaconda2-2.4.1-Windows-x86_64.exe	08-Dec-2015 21:00	371393960
Anaconda2-2.5.0-Linux-x86.sh	03-Feb-2016 21:41	346405513
Anaconda2-2.5.0-Linux-x86_64.sh	03-Feb-2016 21:41	409842279
Anaconda2-2.5.0-MacOSX-x86_64.pkg	03-Feb-2016 21:55	385762781
Anaconda2-2.5.0-MacOSX-x86_64.sh	03-Feb-2016 21:41	331485310
Anaconda2-2.5.0-Windows-x86.exe	03-Feb-2016 21:45	310590880
Anaconda2-2.5.0-Windows-x86_64.exe	03-Feb-2016 21:46	365581384
Anaconda2-4.0.0-Linux-x86.sh	29-Mar-2016 16:14	348392297
Anaconda2-4.0.0-Linux-x86_64.sh	29-Mar-2016 16:14	411562823
Anaconda2-4.0.0-MacOSX-x86_64.pkg	29-Mar-2016 16:14	355703551
Anaconda2-4.0.0-MacOSX-x86_64.sh	29-Mar-2016 16:14	304288480
Anaconda2-4.0.0-Windows-x86.exe	29-Mar-2016 16:15	294659856
Anaconda2-4.0.0-Windows-x86_64.exe	29-Mar-2016 16:14	350807856
Anaconda2-4.1.0-Linux-x86.sh	28-Jun-2016 16:28	340190685
Anaconda2-4.1.0-Linux-x86_64.sh	28-Jun-2016 16:28	418188731
Anaconda2-4.1.0-MacOSX-x86_64.pkg	28-Jun-2016 16:28	360909420
Anaconda2-4.1.0-MacOSX-x86_64.sh	28-Jun-2016 16:28	309460309
Anaconda2-4.1.0-Windows-x86.exe	28-Jun-2016 16:28	298958864
Anaconda2-4.1.0-Windows-x86_64.exe	28-Jun-2016 16:28	356677104
Anaconda2-4.1.1-Linux-x86.sh	08-Jul-2016 16:19	340385173
Anaconda2-4.1.1-Linux-x86_64.sh	08-Jul-2016 16:19	419038579
Anaconda2-4.1.1-MacOSX-x86_64.pkg	08-Jul-2016 16:19	361721748
Anaconda2-4.1.1-MacOSX-x86_64.sh	08-Jul-2016 16:20	310125837
Anaconda2-4.1.1-Windows-x86.exe	08-Jul-2016 16:20	299852168
Anaconda2-4.1.1-Windows-x86_64.exe	08-Jul-2016 16:20	357765440

图 1-3　下载 Anaconda2-2.4.1-Windows-x86.exe 的界面

Anaconda2-2.4.1-Windows-x86.exe(32 位)工具中提供了 Python 作经济金融数据分析的丰富资源：包括 Pandas，NumPy，SciPy，Statsmodels，Matplotlib 等一系列的程序包以及 IPython 交互环境。要了解 Python 的其他程序包，可到 https://anaconda.org 网站上去搜索你所需要的包进行安装。

1.5 数据分析工具 Python 的安装

Python 在 Windows 环境中安装有很多版本。如：(1) Anaconda2-2.4.1-Windows-x86.exe(32 位)版本；(2) Anaconda2-2.4.1-Windows-x86.exe(64 位)；(3) 最新的 Anaconda3-4.1.1-Windows-x86.exe(32 位)；(4) Anaconda3-4.1.1-Windows-x86_64.exe (64 位)。本书使用的是 Anaconda2-2.4.1-Windows-x86.exe(32 位)版本。

双击下载的 Anaconda2-4.1.1-Windows-x86 应用程序，即可得到如图 1-4 的界面。

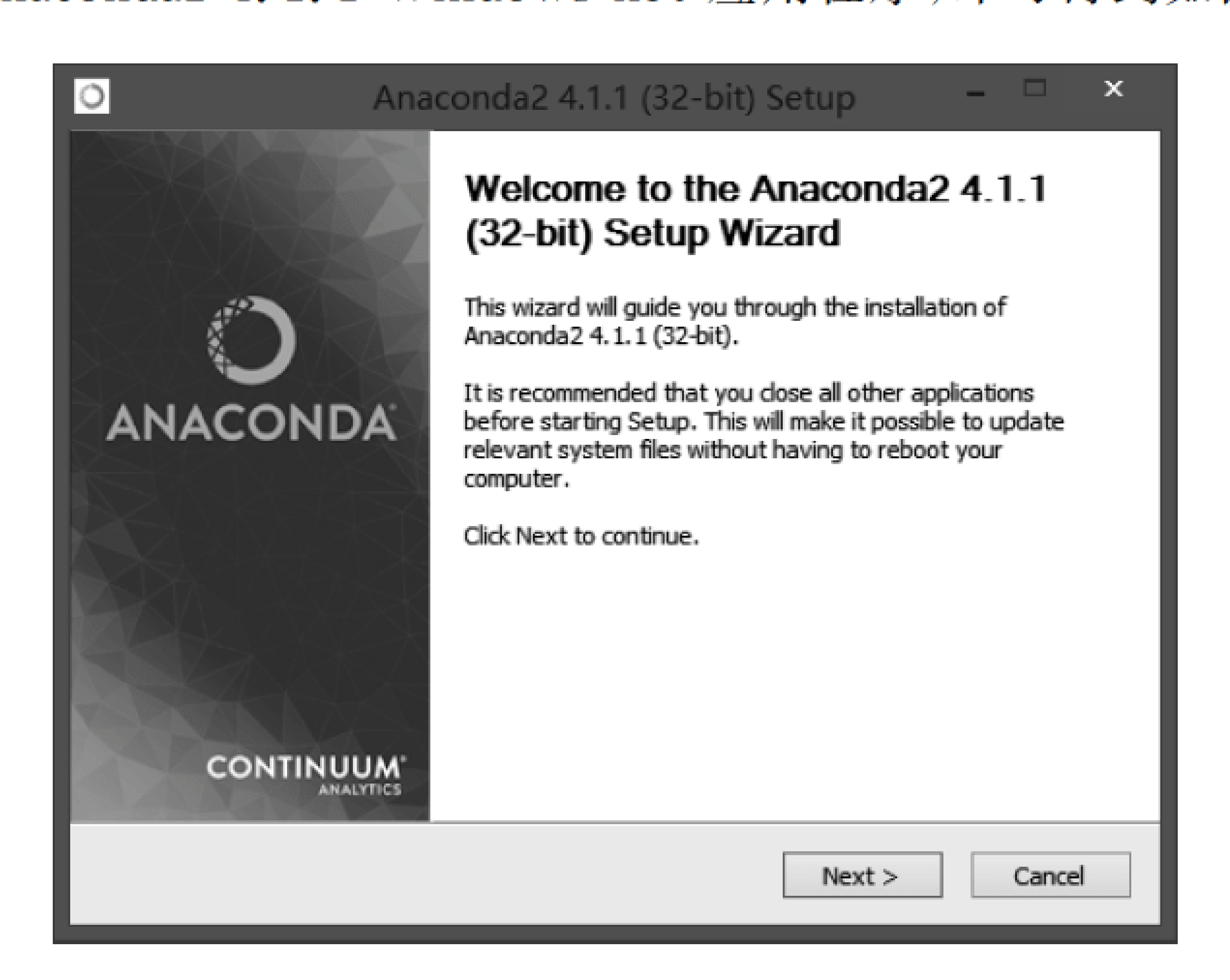


图 1-4 安装界面(1)

在图 1-4 中点击 Next 按钮，得到如图 1-5 所示的界面。

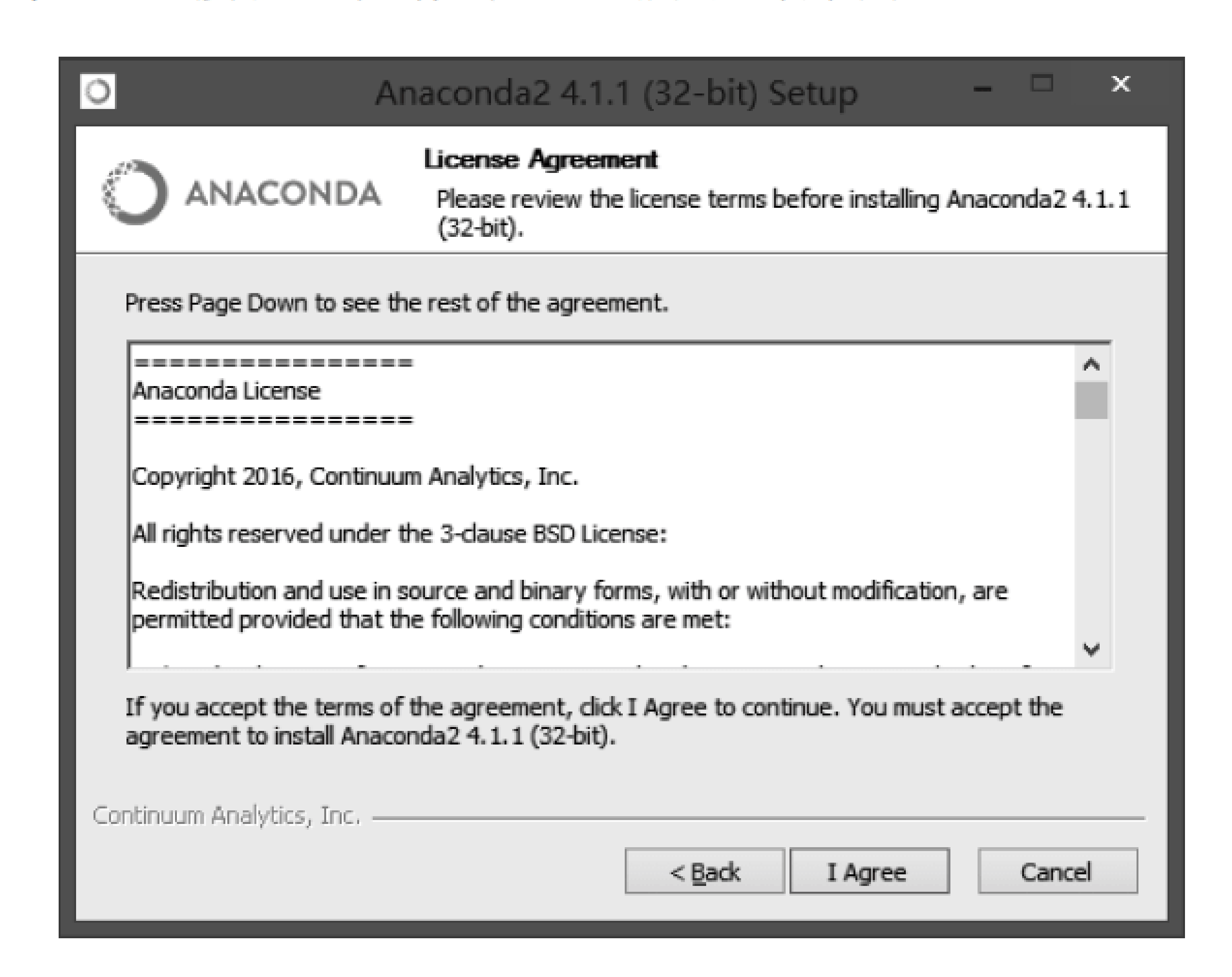


图 1-5 安装界面(2)

在图 1-5 中点击 I Agree 按钮，得到如图 1-6 所示的界面。

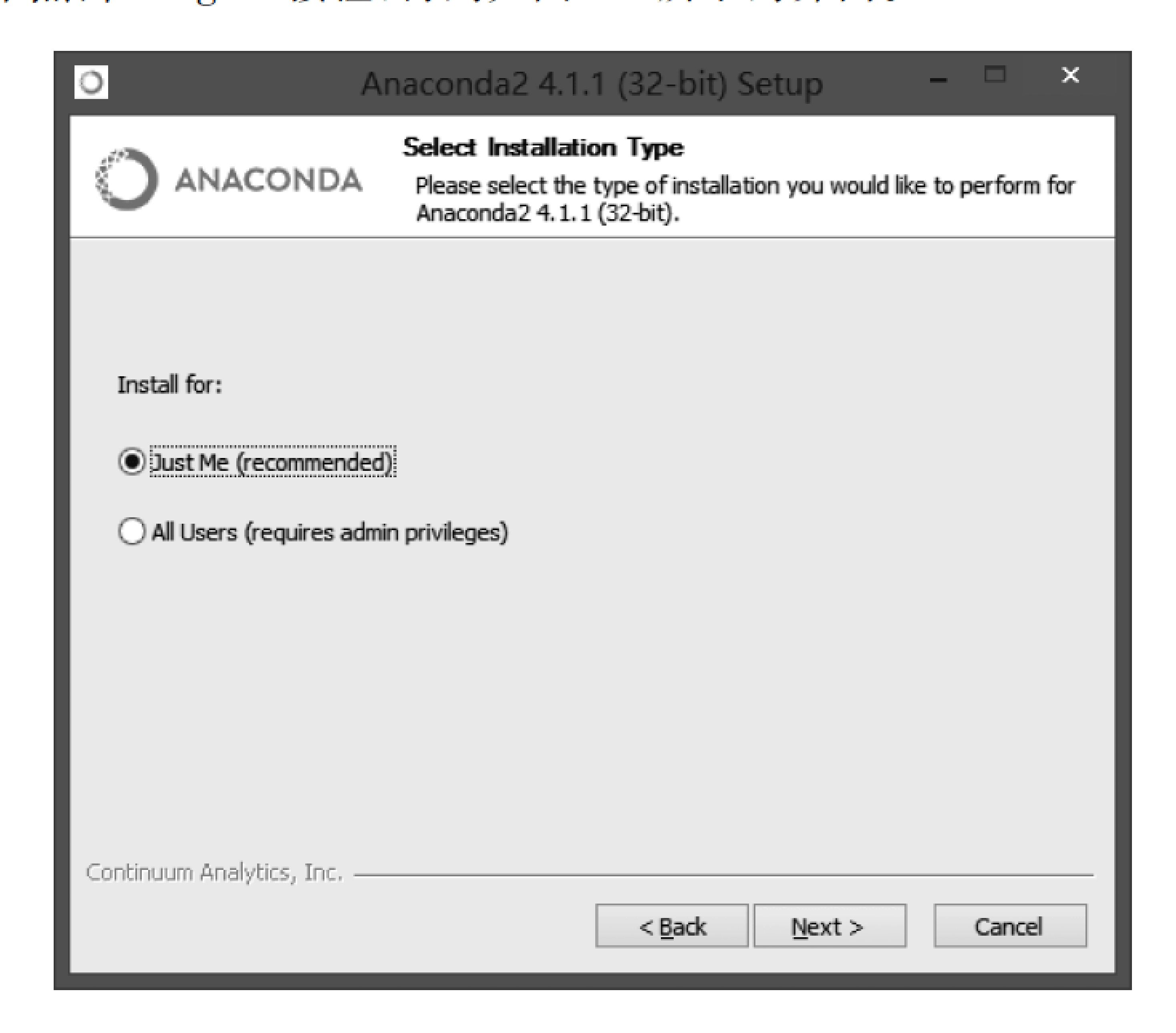


图 1-6　安装向导

点击图 1-6 中的 Next 按钮，得到如图 1-7 所示的界面。

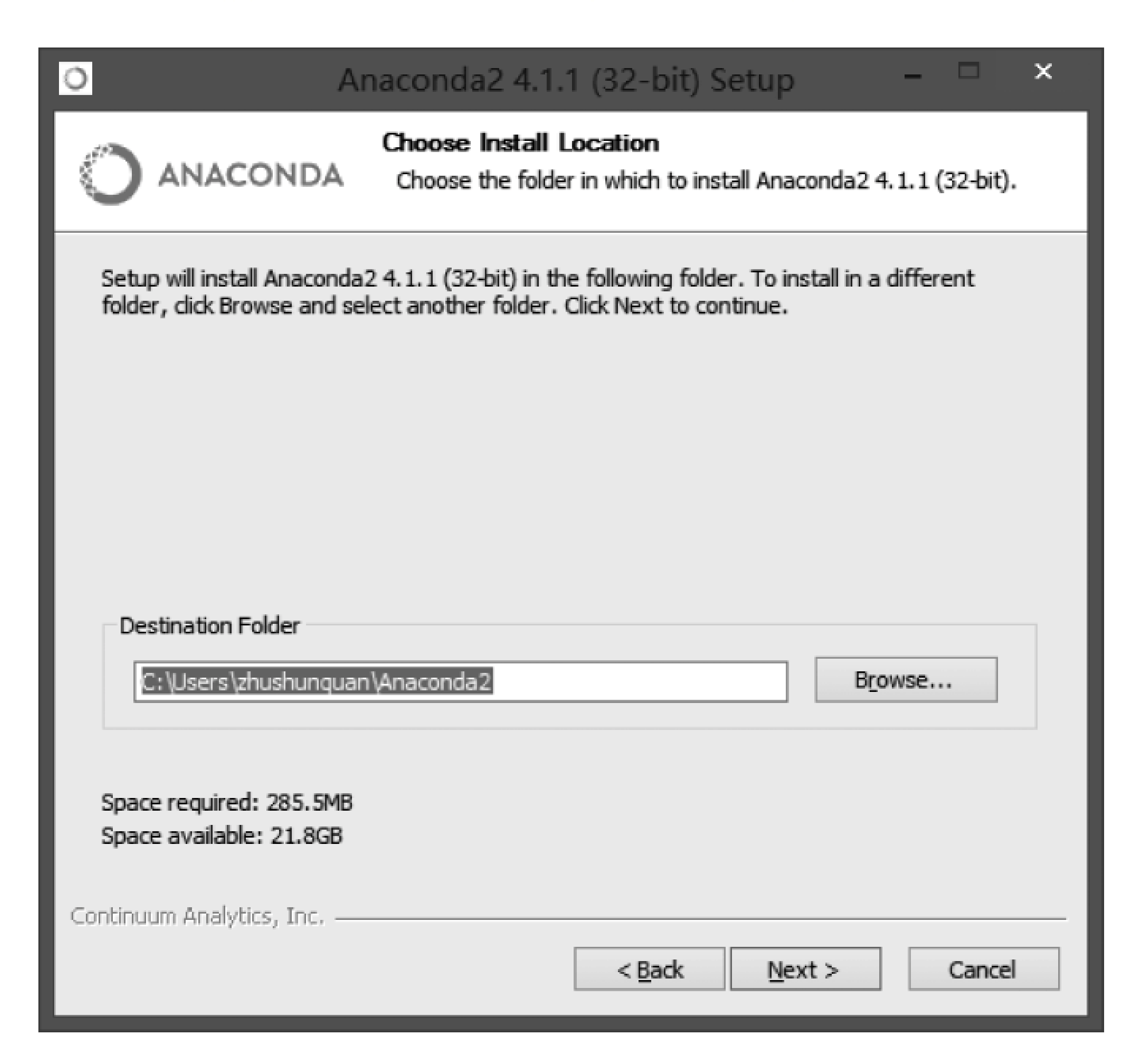


图 1-7　安装向导

点击图 1-7 中的 Next 按钮，即可完成 Python 套装软件的安装，得到如图 1-8 所示的界面。



图 1-8　安装完后界面

1.6　Python 的启动和退出

1.6.1　Python 工具的启动

点击图 1-8 中的 Spider 图标，即可启动 Python 的交互式用户界面。最后得到如图 1-9 所示的界面。Python 是按照问答的方式运行的，即在提示命令符“>>>”后键入命令并回车，Python 就完成一些操作。

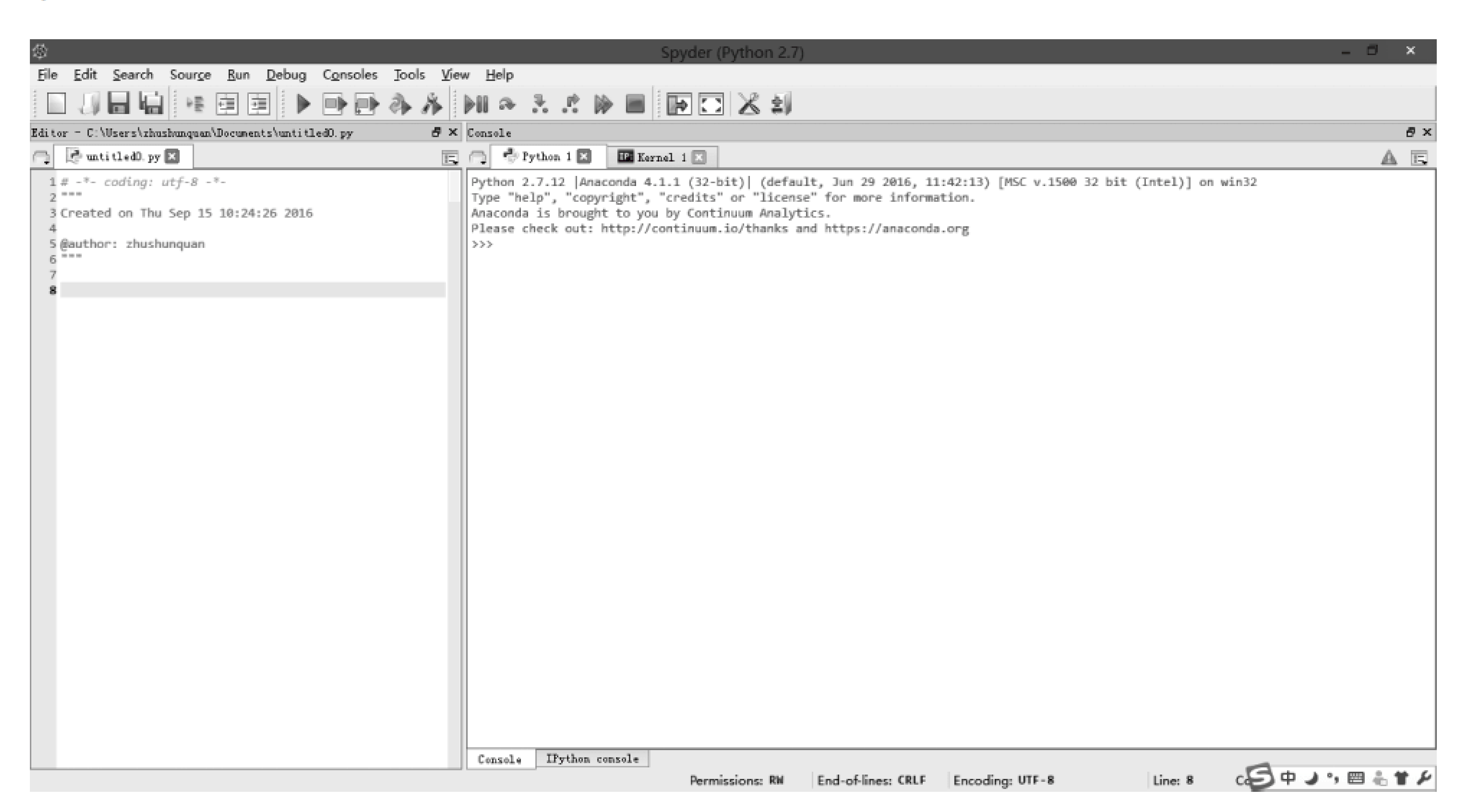


图 1-9　Python 的交互式用户界面

本书经常使用的是如图 1-10 所示的 IPython 控制台界面。

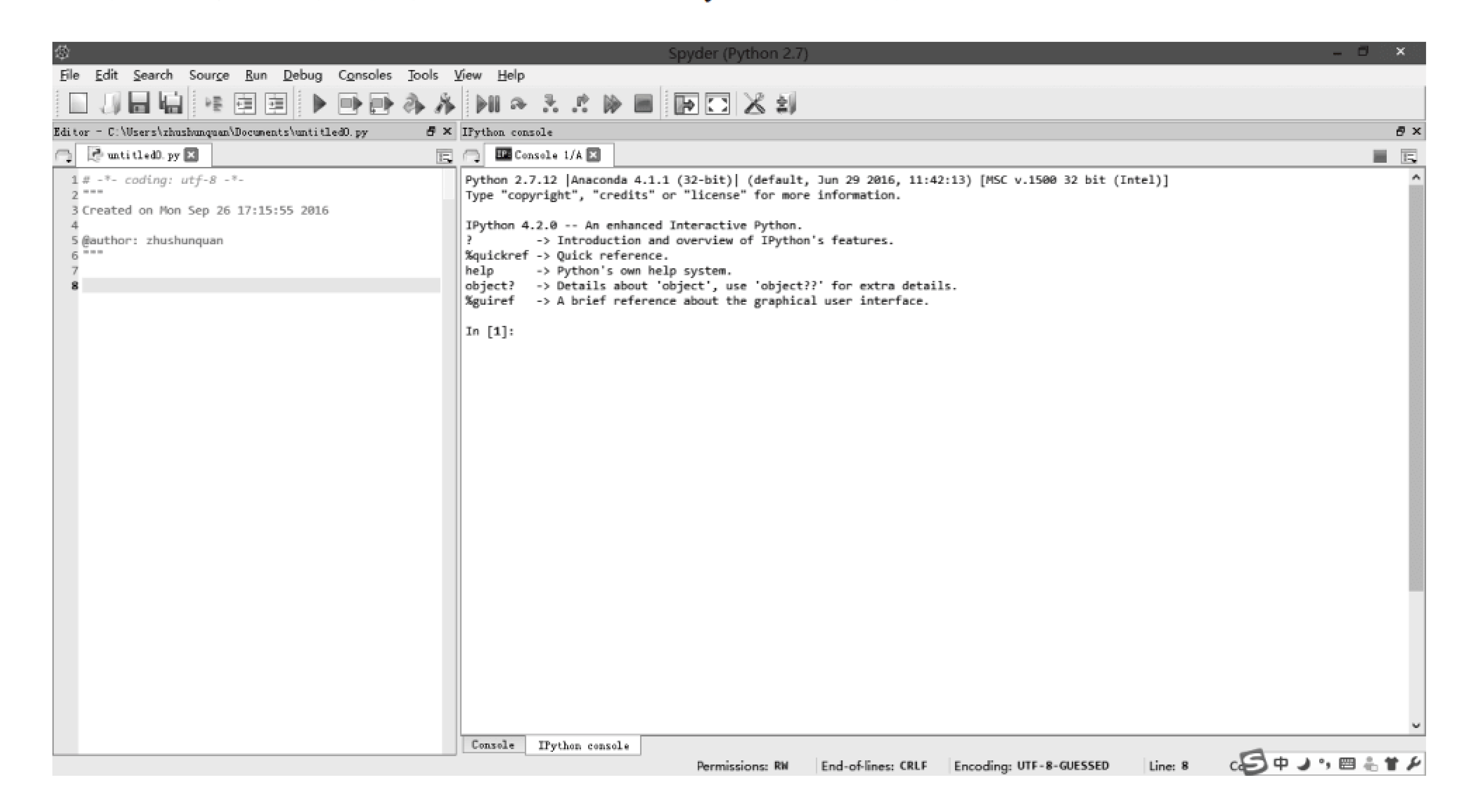


图 1-10　IPython 的交互式用户界面

1.6.2　Python 的退出

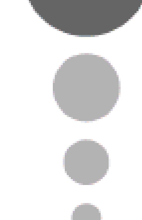
在图 1-9 中的符号“>>>”后同时按 Ctrl 键与 Q 键或点击 Python 交互式用户界面中的“File”下的“Quit”菜单，即可退出 Python 作经济金融数据分析的交互式用户界面。

1.7　Python 数据分析相关的程序包

Python 进行数据分析时，具有获取数据、整理数据、模型计算、数据图形化等功能，相关的 Python 数据分析程序包如表 1-3 所示。

表 1-3　Python 数据分析程序包

程　序　包	简　　介
NumPy	提供数组支持
SciPy	提供矩阵支持，以及矩阵相关的数值计算，优化和统计模块
Pandas	强大、灵活的数据分析和探索工具
Matplotlib	强大的数据可视化工具、作图库
StatsModels	统计建模和计量经济学，包括描述统计、统计模型估计和推断
Scikit-Learn	支持回归、分类、聚类等的强大机器学习库
Keras	深度学习库，用于建立神经网络以及深度学习模型
Gensim	用来做文本主题模型的库，文本挖掘可能用到
Pillow	涉及图片处理
OpenCV	涉及视频处理
GMPY2	涉及高精度运算



1.7.1　Statsmodels 程序包

表 1-3 的 Statsmodels 程序包是 Python 进行统计建模和计量经济学工具，提供一些互补 SciPy 统计计算的功能，包括描述性统计、统计模型估计和推断等。主要功能特性如下。

(1) 线性回归模型：广义最小二乘法（generalized least squares）、普通最小二乘法（ordinary least squares）。

(2) glm：广义线性模型。

(3) discrete：离散变量的回归，基于最大似然估计。

(4) rlm：稳健线性模型。

(5) tsa：时间序列分析模型

(6) nonparametric：非参数估计。

(7) datasets：数据集合。

(8) stats：常用统计检验。

(9) iolib：读 Stata 的.dta 格式，输出 ascii、latex 和 html。

Statsmodels 程序包详细内容参见 https://github.com/statsmodels/statsmodels 主页。

1.7.2　Scikit-Learn 程序包

表 1-3 的 Scikit-Learn 程序包的功能如下。

(1) 所有模型提供的接口有：model.fit()，训练模型，对于监督模型来说是 fit(X,y)，对于非监督模型是 fit(X)。

(2) 监督模型提供的接口有：①model.predict(X_new)，预测新样本；②model.predict_proba(X_new)，预测概率，仅对某些监督模型有用(比如 LR)；③model.score()，得分越高，fit 越好。

(3) 非监督模型提供的接口有：①model.transform()，从数据中学到新的“基空间”；②model.fit_transform()，从数据中学到新的基并将这个数据按照这组“基”进行转换。

1.7.3　Keras 程序包

虽然 Scikit-Learn 足够强大，但是它并没有包含一种强大的模型——人工神经网络。在语言处理、图像识别等领域有着重要的作用。

但要注意的是 Windows 环境下 Keras 的速度会大打折扣。因此，要研究神经网络和深度学习方面的内容，需要在 Linux 下搭建环境。

1.8　Python 数据分析快速入门

1.8.1　数据导入

数据导入是很关键的一步，为了后续的分析，首先需要导入数据。通常来说，数据一般是 csv 格式，就算不是，至少也可以转换成 csv 格式。在 Python 中，操作如下：

```
import Pandas as pd
```

```
# 读取本地数据
df = pd.read_csv('/2glkx/data/al2-1.csv')
#读取网上数据
import Pandas as pd
data_url = "https://raw.githubusercontent.com/alstat/Analysis-with-Programming/master/
2014/Python/Numerical-Descriptions-of-the-Data/data.csv"
df = pd.read_csv(data_url)
```

为了读取本地 csv 文件，我们需要 Pandas 这个数据分析库中的相应模块。其中的 read_csv 函数能够读取本地和 web 数据。

1.8.2 数据变换

有了数据，接下来就是数据变换。统计学家和科学家们通常会在这一步移除分析中的非必要数据。先看看网上读取数据的前 5 行和最后 5 行。操作如下：

```
# Head of the data
print df.head()
    Abra  Apayao  Benguet  Ifugao  Kalinga
0   1243    2934      148    3300    10553
1   4158    9235     4287    8063    35257
2   1787    1922     1955    1074     4544
3  17152   14501     3536   19607    31687
4   1266    2385     2530    3315     8520
# Tail of the data
print df.tail()
    Abra  Apayao  Benguet  Ifugao  Kalinga
74  2505   20878     3519   19737    16513
75 60303   40065     7062   19422    61808
76  6311    6756     3561   15910    23349
77 13345   38902     2583   11096    68663
78  2623   18264     3745   16787    16900
```

对 R 语言程序员来说，上述操作等价于通过 print(head(df))来打印数据的前 6 行，以及通过 print(tail(df))来打印数据的后 6 行。当然 Python 中，默认打印是 5 行，而 R 则是 6 行。因此 R 的代码 head(df, n = 10)，在 Python 中就是 df.head(n = 10)，打印数据尾部也是同样道理。

在 R 语言中，数据列和行的名字通过 colnames 和 rownames 来分别进行提取。在 Python 中，则使用 columns 和 index 属性来提取，操作如下：

```
# Extracting column names
print df.columns
Index([u'Abra', u'Apayao', u'Benguet', u'Ifugao', u'Kalinga'], dtype='object')
# Extracting row names or the index
print df.index
RangeIndex(start=0, stop=79, step=1)
```

数据转置使用 T 方法，操作如下：

```
# Transpose data
print df.T
```

```
          0      1     2      3     4      5     6      7     8      9  \
Abra     1243   4158  1787  17152  1266   5576   927  21540  1039   5424
Apayao   2934   9235  1922  14501  2385   7452  1099  17038  1382  10588
Benguet   148   4287  1955   3536  2530    771  2796   2463  2592   1064
Ifugao   3300   8063  1074  19607  3315  13134  5134  14226  6842  13828
Kalinga 10553  35257  4544  31687  8520  28252  3106  36238  4973  40140

      69     70     71     72     73     74     75     76     77  \
Abra     ...  12763   2470  59094   6209  13316   2505  60303   6311  13345
Apayao   ...  37625  19532  35126   6335  38613  20878  40065   6756  38902
Benguet  ...   2354   4045   5987   3530   2585   3519   7062   3561   2583
Ifugao   ...   9838  17125  18940  15560   7746  19737  19422  15910  11096
Kalinga  ...  65782  15279  52437  24385  66148  16513  61808  23349  68663

            78
Abra      2623
Apayao   18264
Benguet   3745
Ifugao   16787
Kalinga  16900

[5 rows x 79 columns]
```

其他变换，例如排序就是用 sort 属性。现在我们提取特定的某列数据。Python 中，可以使用 iloc 或者 ix 属性，一般使用 ix。例如需数据第一列的前 5 行，操作如下：

```
print df.ix[:, 0].head()
0      1243
1      4158
2      1787
3     17152
4      1266
Name: Abra, dtype: int64
```

要注意的是，Python 的索引是从 0 开始而非 1。为了取出从 11 到 20 行的前 3 列数据，我们有：

```
print df.ix[10:20, 0:3]
     Abra  Apayao  Benguet
10    981    1311     2560
11  27366   15093     3039
12   1100    1701     2382
13   7212   11001     1088
14   1048    1427     2847
15  25679   15661     2942
16   1055    2191     2119
17   5437    6461      734
18   1029    1183     2302
19  23710   12222     2598
20   1091    2343     2654
```

上述命令相当于 df.ix[10:20, ['Abra', 'Apayao', 'Benguet']]。

为了舍弃数据中的列，如列 1(Apayao)和列 2(Benguet)，可使用 drop 属性，操作如下：

```
print df.drop(df.columns[[2, 3]], axis = 1).head()
     Abra  Ifugao  Kalinga
0    1243    3300    10553
1    4158    8063    35257
2    1787    1074     4544
3   17152   19607    31687
4    1266    3315     8520
```

axis 参数告诉函数到底舍弃列还是行。如果 axis 等于 0,那么就舍弃行。

1.8.3 统计描述

下一步就是通过 describe 属性,对数据的统计特性进行描述:

```
print df.describe()
               Abra        Apayao     Benguet        Ifugao       Kalinga
count     79.000000     79.000000    79.000000     79.000000     79.000000
mean   12874.379747  16860.645570  3237.392405  12414.620253  30446.417722
std    16746.466945  15448.153794  1588.536429   5034.282019  22245.707692
min      927.000000    401.000000   148.000000   1074.000000   2346.000000
25 %    1524.000000   3435.500000  2328.000000   8205.000000   8601.500000
50 %    5790.000000  10588.000000  3202.000000  13044.000000  24494.000000
75 %   13330.500000  33289.000000  3918.500000  16099.500000  52510.500000
max    60303.000000  54625.000000  8813.000000  21031.000000  68663.000000
```

1.8.4 假设检验

在 Python 中,有一个很好的统计推断包,就是 SciPy 里面的 Stats。ttest_1samp 实现了单样本 t 检验。因此,如果想检验数据 Abra 列的稻谷产量均值,通过零假设,这里我们假定总体稻谷产量均值为 15000,我们有:

```
from SciPy import stats as ss
# Perform one sample t-test using 1500 as the true mean
print ss.ttest_1samp(a = df.ix[:, 'Abra'], popmean = 15000)
Ttest_1sampResult(statistic = -1.1281738488299586, pvalue = 0.26270472069109496)
```

返回下述值组成的元组。

t:浮点或数组类型;

t 统计量;

prob:浮点或数组类型;

two-tailed p-value:双侧概率值。

通过上面的输出,可以看到 p 值是 0.267,远大于 α(等于 0.05),因此没有充分的证据说平均稻谷产量不是 15000。将这个检验应用到所有的变量,同样假设均值为 15000,我们有:

```
print ss.ttest_1samp(a = df, popmean = 15000)
Ttest_1sampResult(statistic = array([ - 1.12817385, 1.07053437, - 65.81425599, - 4.564575,
6.17156198]),pvalue = array([2.62704721e - 01, 2.87680340e - 01, 4.15643528e - 70, 1.83764399e - 05,
2.82461897e - 08]))
```

上述输出结果的第一个数组是 t 统计量,第二个数组则是相应的 p 值。

1.8.5　可视化

Python 中有许多可视化模块,最流行的是 Matpalotlib 库。可选择功能更强的 seaborn 模块。

```
# Import the module for plotting
import matplotlib.pyplot as plt
plt.show(df.plot(kind = 'box'))
```

得到如图 1-11 所示的图形。

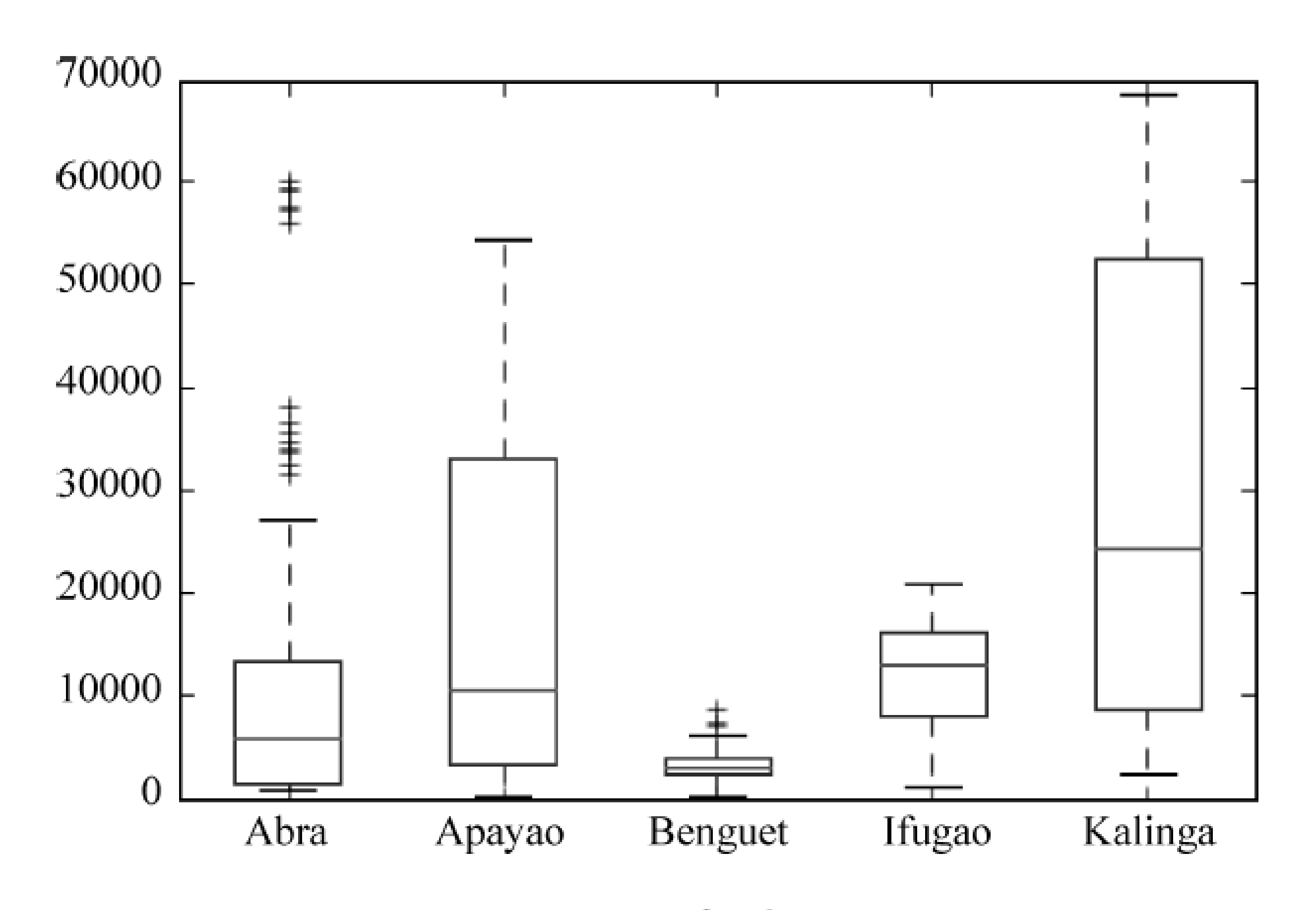


图 1-11　盒型图

现在,我们可以用 Pandas 模块中集成 R 的 ggplot 主题来美化图表。要使用 ggplot,我们只需要在上述代码中多加一行。

```
import matplotlib.pyplot as plt
pd.options.display.mpl_style = 'default'
# Sets the plotting display theme to ggplot2
df.plot(kind = 'box')
```

这样我们就得到如图 1-12 所示的图形。

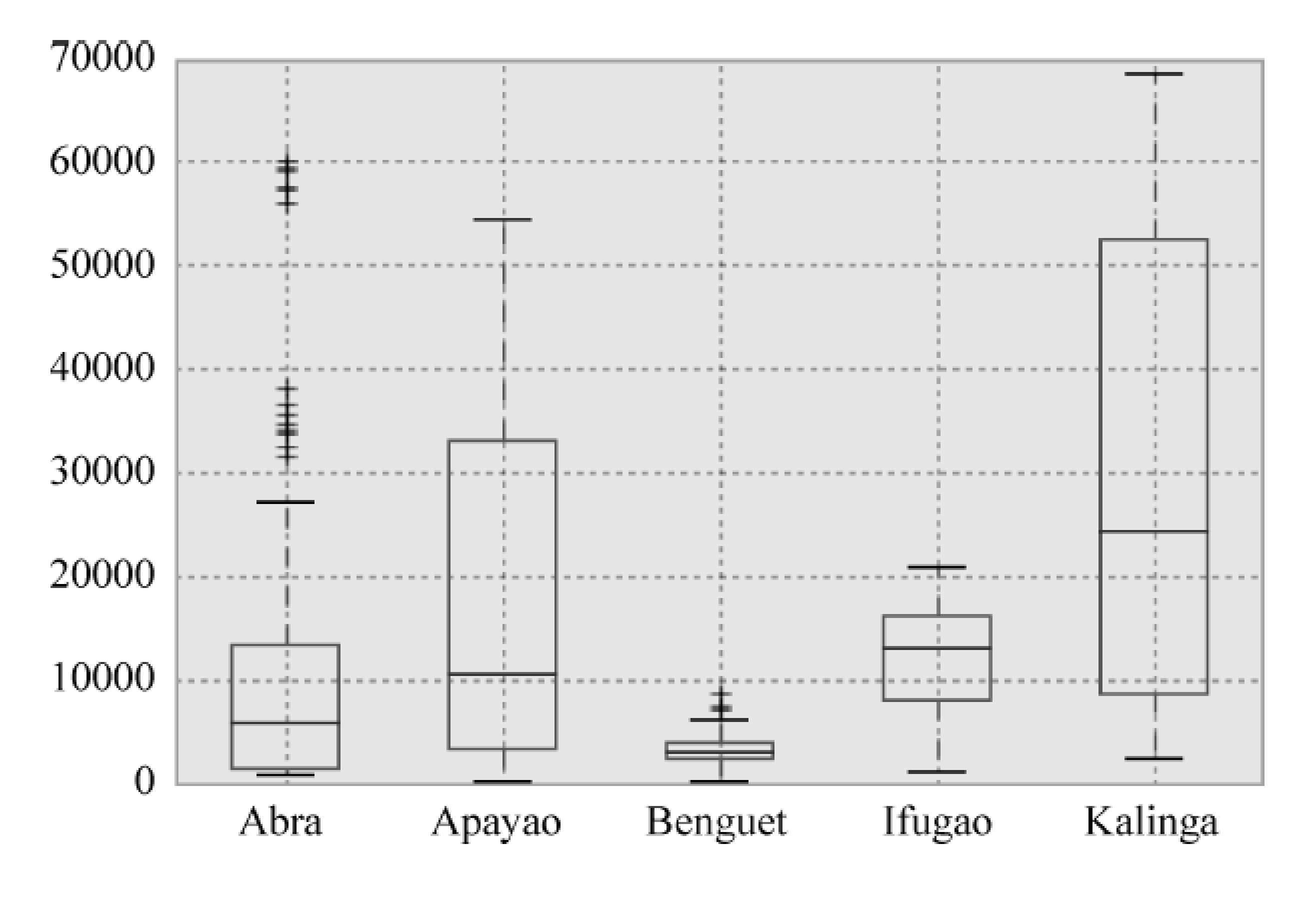


图 1-12　盒型图

可见比 Matplotlib. pyplot 主题简洁太多。下面再引入功能更强大的 seaborn 模块，该模块是一个统计数据可视化库。因此我们有：

```
# Import the seaborn library
import seaborn as sns
# Do the boxplot
plt.show(sns.boxplot(df))
```

可得到如图 1-13 所示的图形。

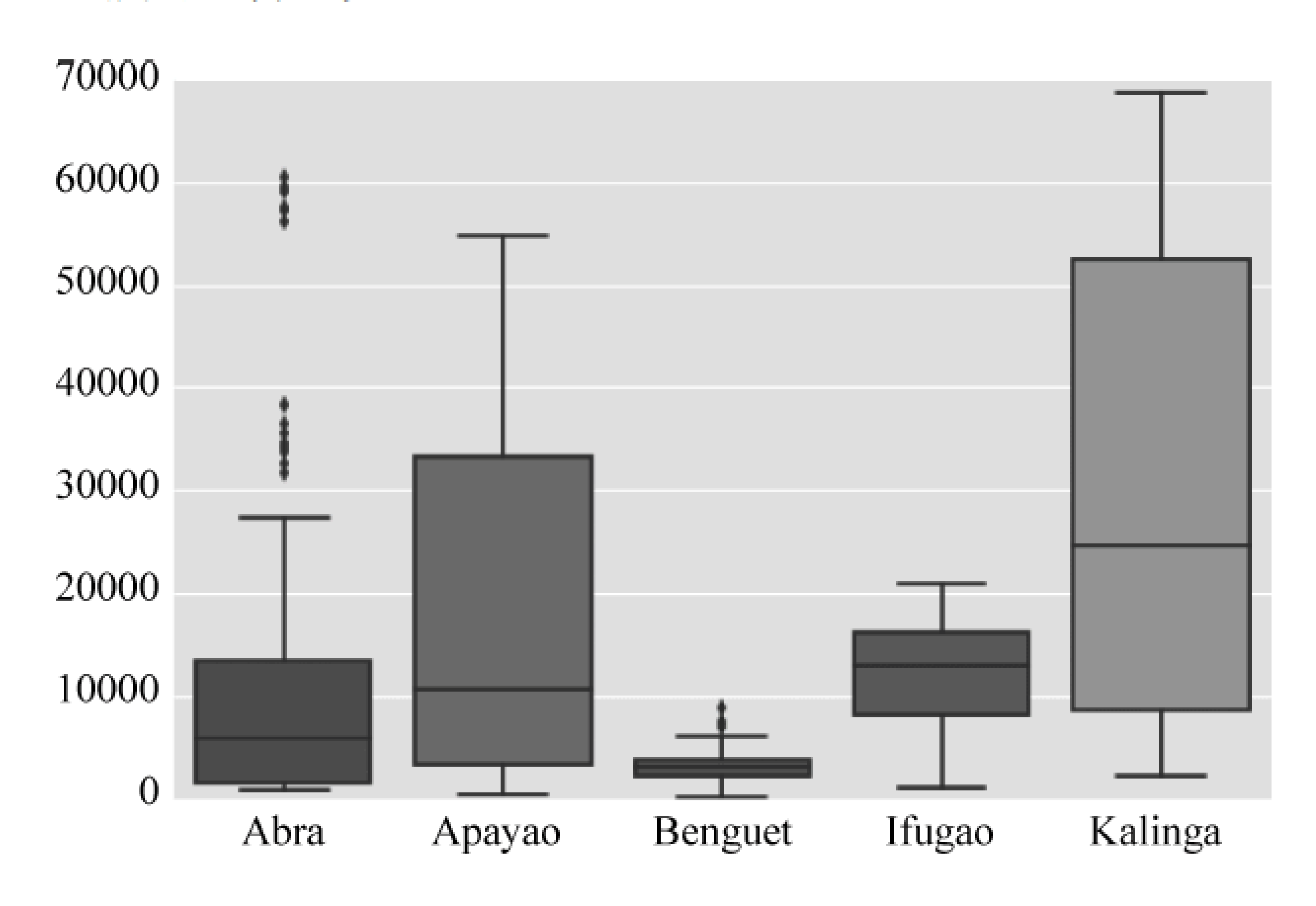


图 1-13　盒型图

1.8.6　创建自定义函数

在 Python 中，我们使用 def 函数来实现一个自定义函数。例如，如果我们要定义一个两数相加的函数，如下即可。

```
def add_2int(x, y):
    return x + y
print add_2int(2, 2)
4
```

练　习　题

1. 简述经济金融数据的类型、来源。
2. 简述经济金融数据分析的常用工具。
3. Python 与 R、Stata、Matlab、SAS、SPSS、EViews 等数据分析工具有何区别？
4. 在网址：https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/下载最新 Python 工具到你指定的目录，并安装到你指定的目录，并启动 Python 软件，然后退出。

第2章 Python 数据分析程序包应用基础

Python 发展至今，已经有越来越多的人使用 Python 进行科技研究，NumPy、SciPy、Pandas 程序包是 Python 最重要的三个高性能数据分析和科学计算的基础包。因此，本章对这三个包应用基础先做一个简单介绍。

2.1 Python 数据分析的 NumPy 应用基础

2.1.1 数组(ndarray)

ndarray(以下简称数组)是 NumPy 的数组对象，需要注意的是，它是同构的，也就是说，其中的所有元素必须是相同的类型。其中，每个数组都有一个 shape 和 dtype。

shape 是数组的形状，例如：

```
import NumPy as np
from NumPy.random import randn
arr = randn(12).reshape(3, 4)
arr
Out[1]:
array([[  0.34640208, -0.99242853, -0.56409477,  0.93549739],
       [-0.58249952, -0.70841307,  0.56573421, -0.09397698],
       [-0.17211688, -0.99705251,  0.22227907,  1.41410709]])
arr.shape
Out[2]: (3, 4)
```

其中(3，4)即代表 arr 是 3 行 4 列的数组，其中 dtype 为 float64。

表 2-1 中的函数可以用来创建数组。

表 2-1　创建数组的函数

函数名称	作　用
array	将输入数据转换为 ndarray，类型可制定也可默认
asarray	将输入转换为 ndarray
arange	类似内置 range
ones、ones_like	根据形状创建一个全 1 的数组、后者可以复制其他数组的形状
zeros、zeros_like	类似上面，全 0
empty、empty_like	创建新数组、只分配空间
eye、identity	创建对角线为 1 的对角矩阵

2.1.2 数组的转置和轴对称

转置是多维数组的基本运算之一。可以使用.T 属性或者 transpose()来实现。.T 就是进行轴对换,而 transpose 则可以接收参数进行更丰富的变换。

```
arr = np.arange(6).reshape((2,3))
print arr
[[0 1 2]
 [3 4 5]]

print arr.T
[[0 3]
 [1 4]
 [2 5]]
arr = np.arange(24).reshape((2,3,4))
print arr
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
print arr.transpose((0,1,2))
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

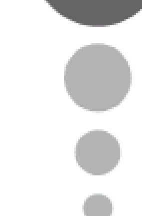
2.1.3 数组的运算

大小相等的数组之间做任何算术运算都会将运算应用到元素级别。

```
arr = np.arange(9).reshape(3, 3)
print arr
[[0 1 2]
 [3 4 5]
 [6 7 8]]

print arr * arr
[[ 0  1  4]
 [ 9 16 25]
 [36 49 64]]

print arr + arr
[[ 0  2  4]
```




```
 [ 6  8 10]
 [12 14 16]]

print arr * 4

[[ 0  4  8]
 [12 16 20]
 [24 28 32]]
```

在 NumPy 的简单计算中，ufunc 通用函数是对数组中的数据执行元素级运算的函数。例如：

```
arr = np.arange(6).reshape((2,3))
print arr
[[0 1 2]
 [3 4 5]]
print np.square(arr)
[[ 0  1  4]
 [ 9 16 25]]
```

类似的函数还有：

abs，fabs，sqrt，square，exp，log，sign，ceil，floor，rint，modf，isnan，isfinite，isinf，cos，cosh，sin，sinh，tan，tanh，add，subtract，multiply，power，mod，equal，等等。

2.2　Python 数据分析的 SciPy 应用基础

SciPy 以 NumPy 为基础，提供了应用更加广泛的科学计算工具。它有着优秀的函数库，主要包括：

(1) 线性代数；

(2) 数值积分；

(3) 插值；

(4) 优化；

(5) 随机数生成；

(6) 信号处理；

(7) 图像处理；

(8) 其他。

与 NumPy 一样，SciPy 有着稳定、成熟、应用广泛的数值运算库。许多 SciPy 函数仅仅是给诸如 LAPACK、BLAS 这样的 Fortran 数值计算工业标准库提供了接口。在本书中，仅仅讨论一些常用基础的函数和特性，不做深入讨论。

2.2.1　SciPy 与 NumPy

由于 SciPy 以 NumPy 为基础，那么 import SciPy 的同时便 import 了 NumPy 库。

所以，我们经常在 SciPy 函数的初始化文件中看到如下代码：

```
from NumPy import *
```

```
from NumPy.random import rand, randn
from NumPy.fft import fft, ifft
from NumPy.lib.scimath import *
del linalg
```

SciPy 的函数主要位于以下子库中：

```
SciPy.optimize   SciPy.integrate SciPy.stats
```

所以这些子库需要分别 import，比如：

```
import SciPy.optimize
from SciPy.integrate import quad
```

尽管 SciPy 已经 import 了 NumPy，但是标准的数值计算程序经常这样引用 NumPy：

```
import NumPy as np
```

2.2.2 统计子库 SciPy.stats

SciPy.stats 的主要功能有以下几方面：

(1) 数值随机变量对象(包括密度分布函数，累积分布函数，样本函数等)；

(2) 一些估计方法；

(3) 一些测试方法。

1. 随机变量与分布

考虑 beta 函数，NumPy 中提供了获取随机变量的样本的方法：

```
import NumPy as np
np.random.beta(5, 5, size=3)
Out[28]: array([ 0.40989776,  0.55822037,  0.33350581])
```

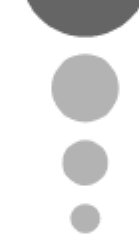
只不过 np.random.beta(a,b)是根据下面函数得到的：

$$f(x,a,b)=\frac{x^{a-1}(1-x)^{b-1}}{\int_0^1 u^{a-1}(1-u)^{b-1}du},\quad 0\leqslant x\leqslant 1$$

为了获取更多 beta 分布的特性，我们经常需要使用 SciPy.stats。

```
import NumPy as np
from SciPy.stats import beta
from matplotlib.pyplot import hist, plot, show
q = beta(5, 5)          # Beta(a, b), with a = b = 5 q是一个对象
obs = q.rvs(2000)       # 2000 observations 获得 2000 个样本
hist(obs, bins=40, normed=True)
grid = np.linspace(0.01, 0.99, 100)
plot(grid, q.pdf(grid), 'k-', linewidth=2)
show()
```

执行上面命令后，可以得到如图 2-1 所示的图形。



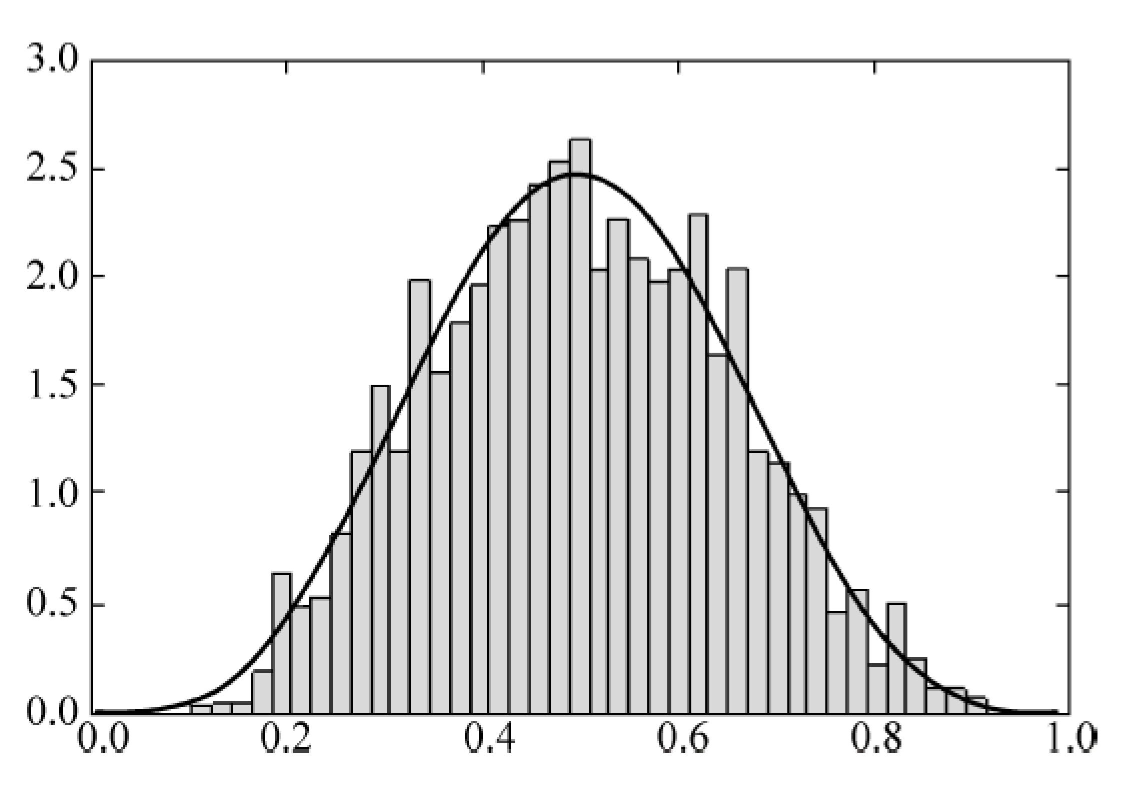


图 2-1　随机分布图

继续输入如下命令：

```
q.cdf(0.4)              # Cumulative distribution function  累积密度函数
Out[31]: 0.26656768000000003

q.pdf(0.4)              # Density function  密度函数
Out[32]: 2.0901888000000013

q.ppf(0.8)              # Quantile (inverse cdf) function
Out[33]: 0.63391348346427079

q.mean()
Out[34]: 0.5
```

通用的用于创建随机变量对象的语法为：

```
identifier = SciPy.stats.distribution_name(shape_parameters)
```

distributon_name 可以在 http://docs.SciPy.org/doc/SciPy/reference/stats.html 中找到。

distributon_name 有两个关键参数 loc 和 scale。

```
identifier = SciPy.stats.distribution_name(shape_parameters, 'loc=c', 'scale=d')
```

这是用来做线性变换，产生的 Y，Y = c+d*X。

这里还有另一种方法生成与上一种方法一样的随机变量，命令如下：

```
import NumPy as np
from SciPy.stats import beta
from matplotlib.pyplot import hist, plot, show
obs = beta.rvs(5, 5, size=2000) # 2000 observations
hist(obs, bins=40, normed=True)
grid = np.linspace(0.01, 0.99, 100)
plot(grid, beta.pdf(grid, 5, 5), 'k-', linewidth=2)
show()
```

执行上面命令后，可以得到如图 2-2 所示的图形。

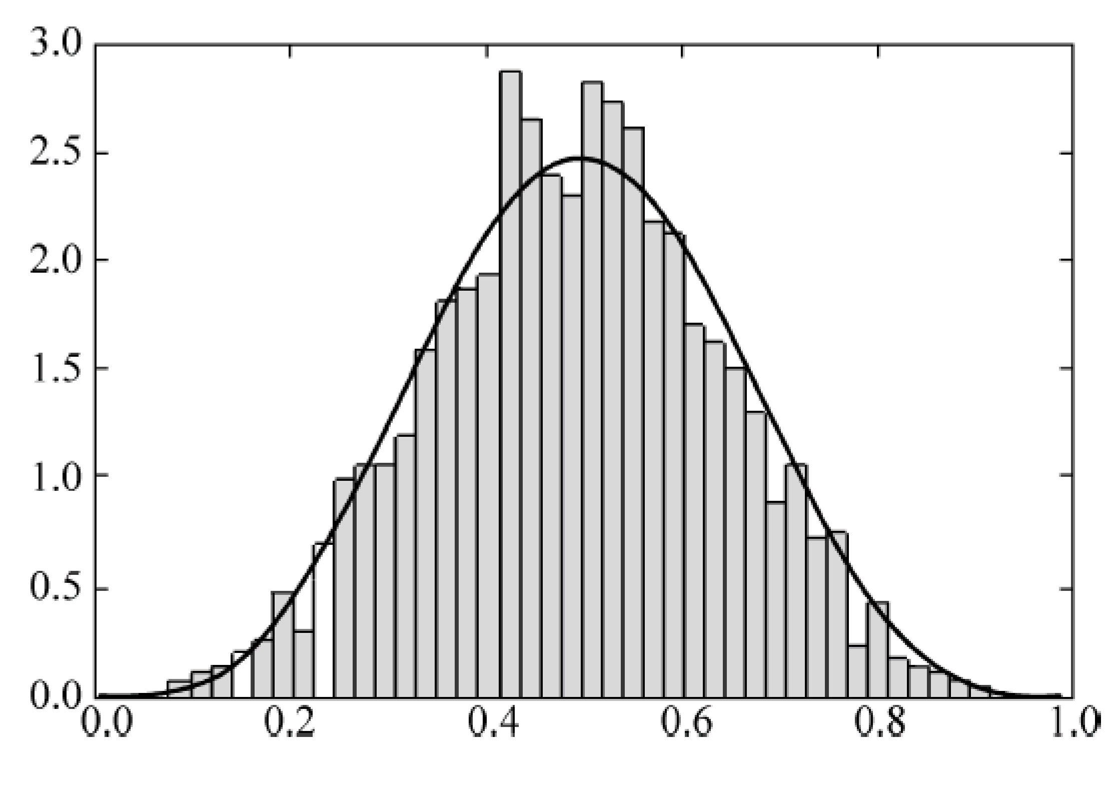


图 2-2　随机分布图

2. 线性回归

```
from SciPy.stats import linregress
x = np.random.randn(200)
y = 2 * x + 0.1 * np.random.randn(200)
gradient, intercept, r_value, p_value, std_err = linregress(x, y)
gradient, intercept
Out[36]: (1.99693537420583, -0.0038124974324087318)
```

3. 求根与稳定点

(1) 稳定点

已知连续函数 $f(x)$ ，则函数 $f(x)$的稳定点为 x_0，使得条件 $f(x_0)=x_0$ 成立。例如以下函数 $f(x)$：

$$f(x) = \sin\left(4\left(x-\frac{1}{4}\right)\right)+x+x^{20}-1$$

(2) 一元函数求根

如果我们要求方程 $f(x)=0$ 的根的话，可以利用 SciPy.optimize 的二分法 bisect。命令如下：

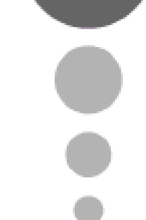
```
from SciPy.optimize import bisect
f = lambda x: np.sin(4 * (x - 0.25)) + x + x**20 - 1
bisect(f, 0, 1)
Out[37]: 0.4082935042797544
```

当然，在数值分析中还有一种常用方法 Newton-Raphson 算法。这一方法利用函数曲线的切线逐渐逼近零点。如果函数的形态很好，那么此方法比 bisect 更快。如果函数的形态不好，那么它比 bisect 慢。这主要是由于此方法依赖于求导运算。

在实际求根过程中，常常采用混合方法，先用一种方法求解，如果速度不快，则寻找另外更好的方法，加速求解。

在 SciPy 中，混合方法是 brentq：

```
from SciPy.optimize import brentq
```




```
brentq(f, 0, 1)
Out[38]: 0.40829350427936706

timeit brentq(f, 0, 1)
10000 loops, best of 3: 51.1 μs per loop
```

(2) 多元函数求根问题

常常采用 SciPy. optimize. fsolve：一个 MinPACK 库接口。详细可参见：

http://docs. SciPy. org/doc/SciPy/reference/generated/SciPy. optimize. fsolve. html 网站上的内容。

(3) 稳定点求解

输入如下命令：

```
from SciPy.optimize import fixed_point
fixed_point(lambda x: x * * 2, 10.0)   # 10.0 is an initial guess
Out[40]: array(1.0)
```

fixed_point 函数仅限于一元函数稳定点问题,因为它仅仅是调用了 brentq 函数。

4. 优化问题

(1) 一元函数最小与最大值

一元函数最小与最大值实例可以通过如下命令来说明：

```
from SciPy.optimize import fminbound
fminbound(lambda x: x * * 2, - 1, 2)   # Search in [ - 1, 2]
Out[41]: 0.0
```

(2) 多元函数最小与最大值

多元函数局部优化：有以下几个函数：

```
minimize, fmin, fmin_powell, fmin_cg, fmin_bfgs, and fmin_ncg。
```

多元函数受限局部优化：fmin_l_bfgs_b，fmin_tnc，fmin_cobyla。

由于涉及更多的搜索算法等数学知识就不详细介绍了。具体问题请到官网 http://docs. SciPy. org/doc/SciPy/reference/optimize. html 查询。

5. 积分

单变量积分可以利用 quad 方法实现,实例的命令如下：

```
from SciPy.integrate import quad
integral, error = quad(lambda x: x * * 2, 0, 1)
integral
Out[45]: 0.33333333333333337
```

quad 方法是依靠 gauss-chebyshev 求积公式,切比雪夫多项式参见：

http://en. wikipedia. org/wiki/Clenshaw%E2%80%93Curtis_quadrature

多元函数积分,固定误差积分等可以参见：

http://docs. SciPy. org/doc/SciPy/reference/integrate. html

6. 图像操作

SciPy 的 imread 可以将图像导入 NumPy 数组。命令如下：

```
from SciPy.misc import imread, imsave, imresize
# Read an JPEG image into a NumPy array
img = imread('assets/cat.jpg')
print img.dtype, img.shape  # Prints "uint8 (400, 248, 3)"
```

从 img.shape 可以看出 img 数组是三维 array，依次是红、绿、蓝。千万不要以为图像数组是 2 维数组。

```
img_tinted = img * [1, 0.95, 0.9]
```

将图像数组分别乘以列表[1,0.95,0.9]，表示红色矩阵数值不变，绿色、蓝色矩阵数值分别降为原来的 0.95 倍、0.9 倍。

```
img_tinted = imresize(img_tinted, (300, 300))
```

将原来的 array 改变为正方形 300 * 300pixels。

彻彻底底变成了“矮猫”。

现在保存图像，命令如下：

```
imsave('assets/cat_tinted.jpg', img_tinted)
```

7. 计算两点间距离

先创建三个点：

```
x = np.array([[0, 1], [1, 0], [2, 0]])
```

[0,1],[1,0],[2,0]分别代表三个点。

这里的两点间距离用欧几里得空间的距离表示。命令如下：

```
from SciPy.spatial.distance import pdist, squareform
d = squareform(pdist(x, 'euclidean'))
d
```

得到实对称矩阵如下：

```
Out[51]:
array([[ 0.        ,  1.41421356,  2.23606798],
       [ 1.41421356,  0.        ,  1.        ],
       [ 2.23606798,  1.        ,  0.        ]])
```

而 SciPy.spatial.distance.cdist(A,B,'euclidean')计算的是两个 array 间各个点的欧几里得距离。

8. 线性规划

考虑如下的线性规划问题：

```
Minimize: f = -1 * x[0] + 4 * x[1]
Subject to: -3 * x[0] + 1 * x[1] <= 6
1 * x[0] + 2 * x[1] <= 4
x[1] >= -3
-∞<= x[0] <= ∞
```

对于上述线性规划问题，输入如下 Python 代码：

```
c = [-1, 4]
A = [[-3, 1], [1, 2]]
b = [6, 4]
x0_bounds = (None, None)
x1_bounds = (-3, None)
from SciPy.optimize import linprog
res = linprog(c, A_ub=A, b_ub=b, bounds=(x0_bounds, x1_bounds),
              options={"disp": True})
print(res)
```

得到如下结果：

```
Optimization terminated successfully.
         Current function value: -22.000000
         Iterations: 1
     fun: -22.0
 message: 'Optimization terminated successfully.'
     nit: 1
   slack: array([ 39.,   0.])
  status: 0
 success: True
       x: array([ 10.,  -3.])
```

2.3　Python 数据分析的 Pandas 应用基础

Python Data Analysis Library 或 Pandas 是 Python 的一个数据分析包，最初由 AQR Capital Management 于 2008 年 4 月开发，并于 2009 年底开源出来，目前由专注于 Python 数据包开发的 PyData 开发团队继续开发和维护，属于 PyData 项目的一部分。

Pandas 最初是被作为金融数据分析工具而开发出来，因此，Pandas 为时间序列分析提供了很好的支持。Pandas 的名称来自于面板数据(panel data)和 Python 数据分析(data analysis)。面板数据(panel data)是经济学中关于多维数据集的一个术语，在 Pandas 中也提供了 panel 的数据类型。

Pandas 是基于 NumPy 的一种工具，该工具是为解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。Pandas 提供了大量能使我们快速便捷地处理数据的函数和方法，从而使 Python 成为强大而高效的数据分析环境的重要因素之一。

2.3.1　Pandas 中的数据结构

Pandas 是基于 NumPy 构建的含有更高级数据结构和工具的数据分析包，Pandas 围绕

Series 和 DataFrame 两个核心数据结构展开。Series 和 DataFrame 分别对应于一维的序列和二维的表结构。

（1）Series：一维数组，与 NumPy 中的一维数组 array 类似。二者与 Python 基本的数据结构 list 也很相近，其区别是：list 中的元素可以是不同的数据类型，而 Array 和 Series 中则只允许存储相同的数据类型，这样可以更有效地使用内存，提高运算效率。

（2）Time-Series：以时间为索引的 Series。

（3）DataFrame：二维的表格型数据结构。很多功能与 R 中的 data. frame 类似。可以将 DataFrame 理解为 Series 的容器。下面的内容主要以 DataFrame 为主。

（4）Panel：三维的数组，可以理解为 DataFrame 的容器。

Pandas 约定俗成的导入方法如下：

```
from Pandas import Series,DataFrame
import Pandas as pd
```

1. 一维数组 Series 对象

Series 可以看做一个定长的有序字典。基本任意的一维数据都可以用来构造 Series 对象：

```
s = pd.Series([1,2,3.0,'abc'])
s
Out[6]:
0       1
1       2
2       3
3     abc
dtype: object
```

虽然 dtype：object 可以包含多种基本数据类型，但总感觉会影响性能的样子，最好还是保持单纯的 dtype。

Series 对象包含两个主要的属性：index 和 values，分别为上例中左右两列。因为传给构造器的是一个列表，所以 index 的值是从 0 起递增的整数，如果传入的是一个类字典的键值对结构，就会生成 index-value 对应的 Series；或者在初始化的时候以关键字参数显式指定一个 index 对象：

```
s = pd.Series(data = [1,3,5,7],index = ['a','b','x','y'])
s
Out[8]:
a    1
b    3
x    5
y    7
dtype: int64
s.index
Out[9]: Index([u'a', u'b', u'x', u'y'], dtype = 'object')
s.values
Out[10]: array([1, 3, 5, 7], dtype = int64)
```

Series 对象的元素会严格依照给出的 index 构建，这意味着：如果 data 参数是有键值对的，那么只有 index 中含有的键会被使用；以及如果 data 中缺少响应的键，即使给出 NaN 值，这个键也会被添加。

注意 Series 的 index 和 values 的元素之间虽然存在对应关系，但这与字典的映射不同。index 和 values 实际仍为互相独立的 ndarray 数组。Series 这种使用键值对的数据结构最大的好处在于，Series 间进行算术运算时，index 会自动对齐。另外，Series 对象和它的 index 都含有一个 name 属性：

```
s.name = 'a_series'
s.index.name = 'the_index'
s
Out[13]:
the_index
a     1
b     3
x     5
y     7
Name: a_series, dtype: int64
```

2. 二维表 DataFrame 对象

DataFrame 是一个表格型的数据结构，它含有一组有序的列(类似于 index)，每列可以是不同的值类型(在 NumPy 的数组中 ndarray 只能有一个数据类型 dtype)。基本上可以把 DataFrame 看成是共享同一个 index 的 Series 的集合。

DataFrame 的构造方法与 Series 类似，只不过可以同时接受多条一维数据源，每一条都会成为单独的一列：

```
data = {'state':['GZ','GZ','GZ','CS','CS'],
        'year':[2014,2015,2016,2015,2016],
        'pop':[1.5,1.7,3.6,2.4,2.9]}
df = DataFrame(data)
df
  Out[16]:
   pop state  year
0  1.5    GZ  2014
1  1.7    GZ  2015
2  3.6    GZ  2016
3  2.4    CS  2015
4  2.9    CS  2016
```

虽然参数 data 看起来是个字典，但字典的键并非充当 DataFrame 的 index 的角色，而是 Series 的“name”属性。这里生成的 index 仍是“01234”。较完整的 DataFrame 构造器参数为：DataFrame(data=None,index=None,columns=None)，columns 即“name”：

```
df = DataFrame(data,index=['one','two','three','four','five'],
               columns=['year','state','pop','debt'])
df
      Out[18]:
```

```
       year  state  pop  debt
one    2014    GZ   1.5  NaN
two    2015    GZ   1.7  NaN
three  2016    GZ   3.6  NaN
four   2015    CS   2.4  NaN
five   2016    CS   2.9  NaN
```

从上可以看出，缺失值由 NaN 补上。

下面看一下 index，columns 和索引的类型：

```
df.index
Out[19]: Index([u'one', u'two', u'three', u'four', u'five'], dtype = 'object')
df.columns
Out[20]: Index([u'year', u'state', u'pop', u'debt'], dtype = 'object')
type(df['debt'])
Out[21]: Pandas.core.series.Series
```

DataFrame 面向行和面向列的操作基本是平衡的，任意抽出一列都是 Series。

2.3.2 对象的重新索引

1. Series 对象的重新索引

Series 对象的重新索引通过其.reindex(index=None, ** kwargs)方法实现。 ** kwargs 中常用的参数有两个：method=None，fill_value=np.NaN。

```
ser = Series([4.5,7.2,-5.3,3.6],index = ['d','b','a','c'])
a = ['a','b','c','d','e']
ser.reindex(a)
Out[24]:
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
ser.reindex(a,fill_value = 0)
Out[25]:
a   -5.3
b    7.2
c    3.6
d    4.5
e    0.0
dtype: float64
ser.reindex(a,method = 'ffill')
a   -5.3
b    7.2
c    3.6
d    4.5
e    4.5
dtype: float64
```

```
ser.reindex(a,fill_value = 0,method = 'ffill')
a   -5.3
b    7.2
c    3.6
d    4.5
e    4.5
dtype: float64
```

.reindex()方法会返回一个新对象，其 index 严格遵循给出的参数，method:{'backfill', 'bfill', 'pad', 'ffill', None}参数用于指定插值(填充)方式，当没有给出时，自动用 fill_value 填充，默认为 NaN(ffill = pad，bfill = back fill，分别指插值时向前还是向后取值)。

2. DataFrame 对象的重新索引方法

DataFrame 对象的重新索引方法：.reindex(index=None，columns=None，** kwargs)。仅比 Series 多了一个可选的 columns 参数，用于给列索引。用法与上例类似，只不过插值方法 method 参数只能应用于行，即轴 0。

```
data = {'T':[1,4,7],
        'U':[0,0,0],
        'C':[2,5,8]}
df = DataFrame(data,index = ['a','c','d'])
df
Out[32]:
   C  T  U
a  2  1  0
c  5  4  0
d  8  7  0
state = ['T','U','C']
df.reindex(index = ['a','b','c','d'],columns = state,method = 'ffill')
 Out[36]:
   T  U  C
a  1  0  2
b  1  0  2
c  4  0  5
d  7  0  8
```

2.3.3　删除指定轴上的项

删除指定轴上的项，即删除 Series 的元素或 DataFrame 的某一行(列)的意思，通过对象的.drop(labels，axis=0)方法。

Series 的数据对象如下：

```
ser
Out[38]:
d    4.5
b    7.2
a   -5.3
c    3.6
```

```
dtype: float64
```

DataFrame 的数据对象如下：

```
df
Out[39]:
   C  T  U
a  2  1  0
c  5  4  0
d  8  7  0
```

删除 ser 对象的 c 行：

```
ser.drop('c')
Out[40]:
d     4.5
b     7.2
a    -5.3
dtype: float64
df
```

删除 df 对象的 a 行：

```
df.drop('a')
Out[41]:
   C  T  U
c  5  4  0
d  8  7  0
```

删除 df 对象的 T 列和 U 列：

```
df.drop(['T','U'],axis=1)
Out[48]:
   C
a  2
c  5
d  8
```

.drop()返回的是一个新对象，原对象不会被改变。

2.3.4 索引和切片

Pandas 支持通过 obj[::]的方式进行索引和切片，以及通过布尔型数组进行过滤。不过需要注意，因为 Pandas 对象的 index 不限于整数，所以当使用非整数作为切片索引时，它是末端包含的。

```
foo = pd.Series([4.5,7.2,-5.3,3.6],index=['a','b','c','d'])
Out[52]:
a     4.5
b     7.2
c    -5.3
d     3.6
dtype: float64
```

```
bar = pd.Series([4.5,7.2,-5.3,3.6])
Out[55]:
0    4.5
1    7.2
2   -5.3
3    3.6
dtype: float64
```

删除 foo 第 3,4 行：

```
foo[:2]
Out[56]:
a    4.5
b    7.2
dtype: float64
```

删除 bar 第 3,4 行：

```
bar[:2]
Out[57]:
0    4.5
1    7.2
dtype: float64
foo[:'c']
Out[58]:
a    4.5
b    7.2
c   -5.3
dtype: float64
```

这里 foo 和 bar 只有 index 不同，bar 的 index 是整数序列。可见当使用整数索引切片时，结果与 Python 列表或 NumPy 的默认状况相同；换成'c'这样的字符串索引时，结果就包含了这个边界元素。

另外一个特别之处在于 DataFrame 对象的索引方式，因为它有两个轴向（双重索引）。

可以这么理解：DataFrame 对象的标准切片语法为：.ix[::,::]。ix 对象可以接受两套切片，分别为行（axis=0）和列（axis=1）的方向。

```
df
 Out[61]:
   C  T  U
a  2  1  0
c  5  4  0
d  8  7  0

df.ix[:2,:2]
   Out[62]:
   C  T
a  2  1
c  5  4
df.ix['a','U']
Out[63]: 0
```

而不使用 ix，直接切的情况就特殊了：(1)索引时，选取的是列；(2)切片时，选取的是行。

```
df['U']
Out[65]:
a     0
c     0
d     0
Name: U, dtype: int64
df[:'c']
 Out[66]:
   C  T  U
a  2  1  0
c  5  4  0
df[:2]
   Out[67]:
   C  T  U
a  2  1  0
c  5  4  0
```

使用布尔型数组的情况，注意行与列的不同切法(列切法的“:”不能省)：

```
df['T']>=4
Out[68]:
a     False
c      True
d      True
Name: T, dtype: bool
df[df['T']>=4]
 Out[69]:
   C  T  U
c  5  4  0
d  8  7  0
df.ix[:,df.ix['c']>=4]
Out[70]:
   C  T
a  2  1
c  5  4
d  8  7
```

2.3.5 算术运算和数据对齐

Pandas 最重要的一个功能是，它可以对不同索引的对象进行算术运算。在将对象相加时，结果的索引取索引对的并集。自动的数据对齐在不重叠的索引处引入空值，默认为 NaN。

```
foo = Series({'a':1,'b':2})
foo
Out[72]:
a     1
```

```
b    2
dtype: int64
bar = Series({'b':3,'d':4})
bar
Out[74]:
b    3
d    4
dtype: int64
foo + bar
Out[75]:
a    NaN
b    5.0
d    NaN
dtype: float64
```

DataFrame 的对齐操作会同时发生在行和列上。当不希望在运算结果中出现 NA 值时，可以使用前面 reindex 中提到过的 fill_value 参数，不过为了传递这个参数，就需要使用对象的方法，而不是操作符：df1.add(df2,fill_value=0)。其他算术方法还有：sub()，div()，mul()。

Series 和 DataFrame 之间的算术运算涉及广泛，这里不讲。

2.3.6　函数应用和映射

NumPy 的 ufuncs(元素级数组方法)可用于操作 Pandas 对象。当希望将函数应用到 DataFrame 对象的某一行或列时，可以使用.apply(func, axis=0, args=(), ** kwds)方法。

```
f = lambda x:x.max() - x.min()
df
 Out[77]:
   C  T  U
a  2  1  0
c  5  4  0
d  8  7  0
df.apply(f)
Out[78]:
C    6
T    6
U    0
dtype: int64
df.apply(f,axis = 1)
Out[79]:
a    2
c    5
d    8
dtype: int64
```

2.3.7　排序和排名

Series 的 sort_index(ascending=True)方法可以对 index 进行排序操作，ascending 参

数用于控制升序或降序，默认为升序。

若要按值对 Series 进行排序，当使用.order()方法，任何缺失值默认都会被放到 Series 的末尾。

在 DataFrame 上，.sort_index(axis=0, by=None, ascending=True)方法多了一个轴向的选择参数与一个 by 参数，by 参数的作用是针对某一(些)列进行排序(不能对行使用 by 参数)：

```
df.sort_index(by='U')
  Out[80]:
   C  T  U
a  2  1  0
c  5  4  0
d  8  7  0
df.sort_index(by=['C','T'])
  Out[81]:
   C  T  U
a  2  1  0
c  5  4  0
d  8  7  0
df.sort_index(axis=1)
  C  T  U
a  2  1  0
c  5  4  0
d  8  7  0
```

排名(Series.rank(method='average', ascending=True))的作用与排序的不同之处在于：排名会把对象的 values 替换成名次(从 1 到 n)。这时唯一的问题在于如何处理平级项，方法里的 method 参数就是起这个作用的，它有 4 个值可选：average, min, max, first。

```
ser=Series([3,2,0,3],index=list('abcd'))
ser
Out[84]:
a    3
b    2
c    0
d    3
dtype: int64
ser.rank()
Out[85]:
a    3.5
b    2.0
c    1.0
d    3.5
dtype: float64
ser.rank(method='min')
Out[86]:
a    3.0
b    2.0
c    1.0
```

```
d    3.0
dtype: float64
ser.rank(method = 'max')
Out[87]:
a    4.0
b    2.0
c    1.0
d    4.0
dtype: float64
ser.rank(method = 'first')
Out[88]:
a    3.0
b    2.0
c    1.0
d    4.0
dtype: float64
```

注意在 ser[0]=ser[3]这对平级项上,不同 method 参数表现出的不同名次。

DataFrame 的.rank(axis=0, method='average', ascending=True)方法多了个 axis 参数,可选择按行或列分别进行排名,暂时好像没有针对全部元素的排名方法。

2.3.8 常用的统计方法

Pandas 对象有一些统计方法。它们大部分都属于约简和汇总统计,用于从 Series 中提取单个值,或从 DataFrame 的行或列中提取一个 Series。比如 DataFrame.mean(axis=0, skipna=True)方法,当数据集中存在 NA 值时,这些值会被简单跳过,除非整个切片(行或列)全是 NA,如果不想这样,则可以通过 skipna=False 来禁用此功能。

设置数据如下:

```
data = {'one':[2012,2013,2014,2015,2016],'two':[1.5,1.7,3.6,2.4,2.9]}
df = DataFrame(data)
df
Out[97]:
    one  two
0  2012  1.5
1  2013  1.7
2  2014  3.6
3  2015  2.4
4  2016  2.9
df.mean()
one     3.083333
two    -2.900000
dtype: float64
df.mean(axis = 1)
Out[98]:
one    2014.00
two       2.42
dtype: float64
df.mean(axis = 1,skipna = False)
```

```
Out[99]:
0    1006.75
1    1007.35
2    1008.80
3    1008.70
4    1009.45
dtype: float64
```

常用的统计方法如表 2-2 所示。

表 2-2　常用的统计方法

名　　称	功　　能
count	非 NA 值的数量
describe	针对 Series 或 DF 的列计算汇总统计
min，max	最小值和最大值
argmin，argmax	最小值和最大值的索引位置(整数)
idxmin，idxmax	最小值和最大值的索引值
quantile	样本分位数(0 到 1)
sum	求和
mean	均值
median	中位数
mode	众数
mad	根据均值计算平均绝对离差
var	方差
std	标准差
skew	样本值的偏度(三阶矩)
kurt	样本值的峰度(四阶矩)
cumsum	样本值的累计和
cummin，cummax	样本值的累计最大值和累计最小值
cumprod	样本值的累计积
diff	计算一阶差分(对时间序列很有用)
pct_change	计算百分数变化

2.3.9　处理缺失数据

Pandas 中 NA 的主要表现为 np.nan，另外 Python 内建的 None 也会被当做 NA 处理。处理 NA 的方法有三种：is(not)null，dropna，fillna。

1．is(not)null

这一方法对对象做元素级应用，然后返回一个布尔型数组，一般可用于布尔型索引。

2．dropna

对于一个 Series，dropna 返回一个仅含非空数据和索引值的 Series。

问题在于对 DataFrame 的处理方式，因为一旦 drop 的话，至少要丢掉一行(列)。这里

的解决方式与前面类似,还是通过一个额外的参数:dropna(axis=0,how='any',thresh=None),how 参数可选的值为 any 或者 all。all 仅在切片元素全为 NA 时才抛弃该行(列)。另外一个有趣的参数是 thresh,该参数的类型为整数,它的作用是,比如 thresh=3,会在一行中至少有 3 个非 NA 值时将其保留。

3. fillna

fillna(value=None, method=None, axis=0)中的 value 参数除了基本类型外,还可以使用字典,这样可以实现对不同的列填充不同的值。method 的用法与前面.reindex()方法相同,这里不再赘述。

要注意的是:在 Series 和 DataFrame 对象的方法中,凡是会对数组作出修改并返回一个新数组的,往往都有一个 replace=False 的可选参数。如果手动设定为 True,那么原数组就可以被替换。

练 习 题

1. 举例说明 NumPy 中的矩阵计算。
2. 举例说明 SciPy 中的统计计算。
3. 举例说明 Pandas 中的对象应用。

第3章 Python 数据分析的数据存取

3.1 Python-NumPy 数据存取

在科学计算与数据分析的过程中，往往需要保存一些数据，也经常需要把保存的这些数据加载到程序中，在 Matlab 中我们可以用 save 和 load 函数很方便地实现。类似地，在 Python 中，我们可以用 NumPy. save()和 NumPy. load()函数达到类似的效果，并且还可以用 SciPy. io. savemat()将数据保存为. mat 格式，用 SciPy. io. loadmat()读取. mat 格式的数据，达到可以和 Matlab 进行数据互动的效果。

下面对上述函数分别介绍。

3.1.1 Python-NumPy 数据保存 NumPy. save()

NumPy. save(arg_1，arg_2)需要两个参数，arg_1 是文件名，arg_2 是要保存的数组。例如：

```
import NumPy as np
a = np.mat('1,2,3;4,5,6')
b = np.array([[1,2,3],[4,5,6]])
np.save('a.npy',a)
np.save('b.npy',b)
```

这个时候 Python 的当前工作路径下就会多出 a. npy 和 b. npy 两个文件，当然我们也可以给出具体的路径，例如：np. save('G:/2glkx/data/a. npy'，a)，就把数据保存在 G:/2glkx/data 的目录中。

3.1.2 Python-NumPy 数据读取 NumPy. load()

下面把保存的这两个数据文件导入 Python：

```
data_a = np.load('a.npy')
data_b = np.load('b.npy')
print 'data_a \n',data_a,'\n the type is',type(data_a)
print 'data_b \n',data_a,'\n the type is',type(data_b)
data_a
[[1 2 3]
 [4 5 6]]
 the type is < type 'NumPy.ndarray'>
```

```
data_b
[[1 2 3]
 [4 5 6]]
 the type is < type 'NumPy.ndarray'>
```

我们可以看到这一过程把原本为矩阵的 a 变为数组型了。

如果想同时保存 a、b 到同一个文件，我们可以用 np.savez()函数，具体用法如下：

```
np.savez('ab.npz',k_a = a,k_b = b)
c = np.load('ab.npz')
print c['k_a']
print c['k_b']
```

得到如下的输出结果：

```
[[1 2 3]
 [4 5 6]]
[[1 2 3]
 [4 5 6]]
```

这时的 c 是一个字典，需要通过关键字取出我们需要的数据。

3.2　Python-SciPy 数据存取

Python-SciPy 数据存取的方法如下：

SciPy.io.savemat() 和 SciPy.io.loadmat()。

首先我们用 SciPy.io.savemat()创建.mat 文件，该函数有两个参数，一个文件名和一个包含变量名和取值的字典。

```
import NumPy as np
from SciPy import io
a = np.mat('1,2,3;4,5,6')
b = np.array([[1,1,1],[2,2,2]])
io.savemat('a.mat', {'matrix': a})
io.savemat('b.mat', {'array': b})
```

至此 Python 的当前工作路径下就多了 a.mat 和 b.mat 这两个文件。

3.3　Python-Pandas 的 csv 格式数据文件存取

Python-Pandas 的 csv 格式数据文件的存取，可以通过 p.to_csv()和 pd.read_csv()函数来解决，实例如下：

```
import Pandas as pd
import NumPy as np
a = ['apple','pear','watch','money']
b = [[1,2,3,4,5],[5,7,8,9,0],[1,3,5,7,9],[2,4,6,8,0]]
d = dict(zip(a,b))
d
```

```
p = pd.DataFrame(d)
p
p.to_csv('G:\\2glkx\\data\\IBM.csv')
```

在 Excel 中打开 IBM.csv 数据文件,得到如图 3-1 所示的数据。

```
pd.read_csv('G:\\2glkx\\data\\IBM.csv')
```

IBM

	A	B	C	D	E
1		apple	money	pear	watch
2	0	1	2	5	1
3	1	2	4	7	3
4	2	3	6	8	5
5	3	4	8	9	7
6	4	5	0	0	9

图 3-1　IBM.csv 中的数据

得到如下数据:

```
Out[14]:
   Unnamed: 0  apple  money  pear  watch
0           0      1      2     5      1
1           1      2      4     7      3
2           2      3      6     8      5
3           3      4      8     9      7
4           4      5      0     0      9
```

3.4　Python-Pandas 的 Excel 格式数据文件存取

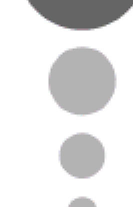
Python-Pandas 的 Excel 格式数据文件的存取,可以通过 pd.read_excel()和 pd.read_csv()函数来解决。实例如下:

先在 G 盘的\\2glkx\\data\\目录下建立一个名为 al3-1.xls 的 Excel 文件。如图 3-2 所示。

al3-1

	A	B	C	D	E	F
1	BH	Z1	Z2	Z3	Z4	K
2	1	7	26	6	60	78.5
3	2	1	29	15	52	74.3
4	3	11	56	8	20	104.3
5	4	11	31	8	47	87.6
6	5	7	52	6	33	95.9
7	6	11	55	9	22	109.2
8	7	3	71	17	6	102.7
9	8	1	31	22	44	72.5
10	9	2	54	18	22	93.1
11	10	21	47	4	18	115.9
12	11	1	40	23	34	83.8
13	12	11	66	9	12	113.3
14	13	10	68	8	12	109.4

图 3-2　Excel 文件



然后通过如下命令来读取 Excel 文件中的数据。

```
import Pandas as pd
import NumPy as np
df = pd.read_excel('G:\\2glkx\\data\\al3-1.xls')
df.head()
```

得到如下数据：

```
Out[23]:
   BH  Z1  Z2  Z3  Z4      K
0   1   7  26   6  60   78.5
1   2   1  29  15  52   74.3
2   3  11  56   8  20  104.3
3   4  11  31   8  47   87.6
4   5   7  52   6  33   95.9
```

3.5　读取并查看数据表列

准备工作完成后，开始读取数据，这里我们使用了一组 Z1 和 Z2 的数据。将这组数据读取到 Python 中并取名为 data。通过 head 函数查看数据表中前 5 行的内容。以下是数据读取和查看的代码和结果。

```
import Pandas as pd
import NumPy as np
#读取数据并创建数据表,名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al3-1.xls'))
#查看数据表前 5 行的内容
data.head()
```

在 data 数据表中，我们将 Z1 设置为自变量 X，将 Z2 设置为因变量 Y。并通过 shape 函数查看了两个变量的行数，每个变量 13 行，这是我们完整数据表的行数。

```
#将 Z1 设为自变量 X
X = np.array(data[['Z1']])
#将 Z2 设为因变量 Y
Y = np.array(data[['Z2']])
#查看自变量和因变量的行数
X.shape,Y.shape
```

3.6　读取 Yahoo 财经网站数据

我们可以使用 Python 的 Pandas 读取 Yahoo 财经网站的数据，代码如下：

```
import Pandas.io.data as web
from Pandas import DataFrame
data_feed = {}
#从 Yahoo 财经网站读取 Apple 和 Facebook 股票 2016/1/1 到 2016/10/1 期间的数据
```

```
data_feed[1] = web.get_data_yahoo('AAPL', '01/1/2016', '10/1/2016')
data_feed[2] = web.get_data_yahoo('FB', '01/1/2016', '10/1/2016')
price = DataFrame({tic: data['Adj Close'] for tic, data in data_feed.iteritems()})
#显示前 5 条记录
price.head()
Out[1]:
                         1           2
Date
2016-01-04  103.586180  102.220001
2016-01-05  100.990380  102.730003
2016-01-06   99.014030  102.970001
2016-01-07   94.835186   97.919998
2016-01-08   95.336649   97.330002
#显示最后 5 条记录
price.tail()
Out[2]:
                         1           2
Date
2016-09-26  112.879997  127.309998
2016-09-27  113.089996  128.690002
2016-09-28  113.949997  129.229996
2016-09-29  112.180000  128.089996
2016-09-30  113.050003  128.270004
```

3.7 读取挖地兔财经网站数据

我们可以使用 Python 的 Pandas 读取挖地兔财经网站数据，代码如下：

```
import tushare as ts            #需先安装 tushare 程序包
#此程序包的安装命令: pip install tushare
import Pandas as pd
import NumPy as np
symbols = [ '600000', '000980', '000981']
#把相对应股票的收盘价按照时间的顺序存入 DataFrame 对象中
data = pd.DataFrame()
data1 = ts.get_hist_data('600000','2016-01-01','2016-10-1')
data1 = data1['close']
data1 = data1[::-1]             #按日期从 2016/1/1 开始到 2016/10/1 结束
data['600000'] = data1
data2 = ts.get_hist_data('000980','2016-01-01','2016-10-1')
data2 = data2['close']
data2 = data2[::-1]
data['000980'] = data2
data3 = ts.get_hist_data('000981','2016-01-01','2016-10-1')
data3 = data3['close']
data3 = data3[::-1]
data['000981'] = data3
data.info()                     #查看数据情况
<class 'Pandas.core.frame.DataFrame'>
```

```
Index: 166 entries, 2016-01-04 to 2016-09-30
Data columns (total 3 columns):
600000      166 non-null float64
000980      106 non-null float64
000981      137 non-null float64
dtypes: float64(3)
memory usage: 4.5+ KB
```

由上可见，三个股票数据的记录不一致，有些股票有 null 值。

```
#清理数据
data = data.dropna()
data.info()
<class 'Pandas.core.frame.DataFrame'>
Index: 106 entries, 2016-04-12 to 2016-09-09
Data columns (total 3 columns):
600000      106 non-null float64
000980      106 non-null float64
000981      106 non-null float64
dtypes: float64(3)
memory usage: 2.9+ KB
```

由上可见，三个股票数据的记录一致，null 值消除。

```
#显示前 5 条
data.head()
Out[13]:
            600000  000980  000981
date
2016-04-12  17.62    6.82    9.20
2016-04-13  17.75    7.50    9.25
2016-04-14  17.80    8.25    9.62
2016-04-15  17.89    9.08    9.55
2016-04-18  17.82    9.99    9.30
#显示最后 5 条
data.tail()
Out[14]:
            600000  000980  000981
date
2016-09-05  16.50   10.10   10.17
2016-09-06  16.42   10.38   10.18
2016-09-07  16.48   10.22   10.19
2016-09-08  16.60   10.30   10.27
2016-09-09  16.55   10.54   10.22
#取列数据
data = data[['600000', '000981']]
data.head()
Out[20]:
            600000  000981
date
2016-04-12  17.62    9.20
2016-04-13  17.75    9.25
```

```
2016-04-14    17.80     9.62
2016-04-15    17.89     9.55
2016-04-18    17.82     9.30
#取 2 行到 4 行的数据
data.ix[1:4]
Out[21]:
              600000   000981
date
2016-04-13    17.75     9.25
2016-04-14    17.80     9.62
2016-04-15    17.89     9.55
#取第 1 行到第 2 行及第 1 列到第 3 列的数据
data.iloc[:2, :3]
Out[24]:
              600000   000981
date
2016-04-12    17.62     9.20
2016-04-13    17.75     9.25
```

3.8 挖地兔 Tushare 财经网站数据保存与读取

Tushare 提供的数据存储模块主要是引导用户将数据保存在本地磁盘或数据库服务器上，便于后期的量化分析和回测使用，以文件格式保存在电脑磁盘的方式上，调用的是 Pandas 本身自带的方法，此处会罗列常用的参数和说明，另外，也会通过实例，展示操作的方法。

(1) 保存为 csv 格式；

(2) 保存为 Excel 格式。

3.8.1 保存为 csv 数据文件

Pandas 的 DataFrame 和 Series 对象提供了直接保存 csv 文件格式的方法，通过参数设定，轻松将数据内容保存在本地磁盘。

常用参数说明：

* path_or_buf：csv 文件存放路径或者 StringIO 对象
* sep:文件内容分隔符，默认为逗号
* na_rep:在遇到 NaN 值时保存为某字符，默认为''空字符
* float_format：float 类型的格式
* columns:需要保存的列，默认为 None
* header:是否保存 columns 名，默认为 True
* index:是否保存 index，默认为 True
* mode:创建新文件还是追加到现有文件，默认为新建
* encoding:文件编码格式
* date_format:日期格式

注：在设定 path 时，如果目录不存在，程序会提示 IOError，请先确保目录已经存在于

磁盘中。

调用方法如下：

```
import tushare as ts
df = ts.get_hist_data('000875') #从网上取数据
#直接保存
df.to_csv('G:/2glkx/data/000875.csv')
#选择数据保存
df.to_csv('G:/2glkx/data/000875.csv',columns = ['open','high','low','close'])
```

2. 读取 csv 数据文件

```
import Pandas as pd
import NumPy as np
df = pd.read_csv('G:/2glkx/data/000875.csv')
df.head()
Out[7]:
   Unnamed: 0  open  high  low  close
0           0  9.78  9.95  9.78   9.89
1           1  9.79  9.83  9.71   9.71
2           2  9.91  9.94  9.77   9.80
3           3  9.61  9.93  9.60   9.86
4           4  9.61  9.75  9.41   9.70
```

追加数据的方式：

某些时候，可能需要将一些同类数据保存在一个大文件中，这时候就需要将数据追加在同一个文件里，简单举例如下：

```
import tushare as ts
import os
filename = 'G:/2glkx/data/bigfile.csv'
for code in ['000875', '600848', '000981']:
    df = ts.get_hist_data(code)
    if os.path.exists(filename):
        df.to_csv(filename, mode = 'a', header = None)
    else:
        df.to_csv(filename)
```

注：如果是不考虑 header，直接 df.to_csv(filename, mode='a')即可，否则，每次循环都会把 columns 名称也 append 进去。

3.8.3　保存为 Excel 文件

Pandas 将数据保存为 Microsoft Excel 文件格式。

常用参数说明：

* excel_writer：文件路径或者 ExcelWriter 对象
* sheet_name：sheet 名称，默认为 Sheet1
* sep：文件内容分隔符，默认为逗号

* na_rep:在遇到 NaN 值时保存为某字符,默认为''空字符
* float_format: float 类型的格式
* columns:需要保存的列,默认为 None
* header:是否保存 columns 名,默认为 True
* index:是否保存 index,默认为 True
* encoding:文件编码格式
* startrow:在数据的头部留出 startrow 行空行
* startcol:在数据的左边留出 startcol 列空列

调用方法如下:

```
import tushare as ts
df = ts.get_hist_data('000875')#直接保存
df.to_excel('G:/2glkx/data/000875.xls')

#设定数据位置(从第 3 行,第 6 列开始插入数据)
df.to_excel('G:/2glkx/data/000875.xls', startrow=2,startcol=5)
```

3.8.4 读取 Excel 数据文件

```
import Pandas as pd
import NumPy as np
df=pd.read_excel('G:/2glkx/data/000875.xls')
df.head()
Out[8]:
         date  open  high  close   low     volume  price_change  p_change  \
0  2016-09-30  9.78  9.95   9.89  9.78  131285.98          0.18      1.85
1  2016-09-29  9.79  9.83   9.71  9.71   98927.06         -0.09     -0.92
2  2016-09-28  9.91  9.94   9.80  9.77   91305.71         -0.06     -0.61
3  2016-09-27  9.61  9.93   9.86  9.60  172743.69          0.16      1.65
4  2016-09-26  9.61  9.75   9.70  9.41  196297.12          0.09      0.94

     ma5   ma10    ma20      v_ma5     v_ma10     v_ma20  turnover
0  9.792  9.791   9.951  138111.91  119221.06  164760.83      0.51
1  9.736  9.789   9.992  143008.49  116914.86  177486.79      0.38
2  9.756  9.808  10.036  146922.54  119919.89  199339.76      0.35
3  9.760  9.809  10.103  144481.56  130811.11  237806.81      0.67
4  9.756  9.845  10.167  123659.19  131849.10  309501.15      0.76
```

练 习 题

1. 按照本章的实例,应用 Python-NumPy 的方法存取数据。
2. 按照本章的实例,应用 Python-SciPy 的方法存取数据。
3. 按照本章的实例,应用 Python-Pandas 的方法存取 csv 和 Excel 格式的数据。



Python 图形的绘制和可视化

4.1 Matplotlib 绘图应用基础

Python 提供了非常多样的绘图功能，可以通过 Python 提供的工具 Matplotlib 绘制二维、三维图形。还有一个 Seaborn 是 Python 中用于创建信息丰富和有吸引力的统计图形库的，它是基于 Matplotlib 的。Seaborn 提供多种功能，如内置主题、调色板、函数和工具，来实现单因素、双因素、线性回归、数据矩阵、统计时间序列等的可视化，以便我们进一步构建更加复杂的可视化。

Matplotlib 库里的常用对象类的包含关系为：Figure -> Axes ->(Line2D，Text，etc.)，一个 Figure 对象可以包含多个子图(Axes)，在 Matplotlib 中用 Axes 对象表示一个绘图区域，可以理解为子图。我们可以使用 subplot()快速绘制包含多个子图的图表，它的调用形式如下：

```
subplot(numRows, numCols, plotNum)
```

subplot 将整个绘图区域等分为 numRows 行乘 numCols 列子区域，然后按照从左到右、从上到下的顺序对每个子区域进行编号，左上的子区域的编号为 1。如果 numRows，numCols 和 plotNum 这三个数都小于 10 的话，可以把它们缩写为一个整数，例如 subplot(323)和 subplot(3，2，3)是相同的。subplot 在 plotNum 指定的区域中创建一个轴对象。如果新创建的轴和之前创建的轴重叠的话，之前的轴将被删除。

这里不可能详细说明 Matplotlib 在绘图方面的所有功能，主要是因为每个绘图函数都有大量的选项，使得图形的绘制十分灵活多变。

Matplotlib 常用的制图功能有：直方图、散点图、曲线标绘图、连线标绘图、箱图、饼图、条形图、点图等，下面我们通过实例来说明 Matplotlib 几种主要图形的绘制方法。

4.2 直方图的绘制

直方图又叫柱状图，是一种统计报告图，由一系列高度不等的纵向条纹或线段表示数据分布的情况，一般用横轴表示数据类型，纵轴表示分布情况。通过绘制直方图，可以较为直观地传递有关数据的变化信息，使数据使用者能够较好地观察数据波动的状态，使数据决策者依据分析结果确定在什么地方需要集中力量改进工作。

例 4-1 为了解某公司雇员的销售和收入情况，我们搜集整理了某公司 10 个雇员的销售和收入有关方面的数据，如表 4-1 所示。试通过绘制直方图来直观显示该公司职员的有关情况。

表 4-1 某公司雇员的销售和收入等情况

EMPID（雇员号）	Gender（性别）	Age（年龄）	Sales（销售）	BMI（体质指数身高和体重）	Income（收入）
EM001	M	34	123	Normal	350
EM002	F	40	114	Overweight	450
EM003	F	37	135	Obesity	169
EM004	M	30	139	Overweight	189
EM005	F	44	117	Overweight	183
EM006	M	36	121	Normal	80
EM007	M	32	133	Obesity	166
EM008	F	26	140	Normal	120
EM009	M	32	133	Normal	75
EM010	M	36	133	Overweight	40

在目录 G:\2glkx\data 下建立 al4-1. xls 数据文件后，导入图形库和数据集的命令如下：

```
import matplotlib.pyplot as plt
import Pandas as pd
import NumPy as np
df = pd.read_excel("G:/2glkx/data/al4 - 1.xls")
# 或者 df = pd.read_excel('G:\\2glkx\\data\\al4 - 1.xls')
df.head()
```

得到如下数据：

```
Out[21]:
   EMPID Gender  Age  Sales         BMI  Income
0  EM001      M   34    123      Normal     350
1  EM002      F   40    114  Overweight     450
2  EM003      F   37    135     Obesity     169
3  EM004      M   30    139  Overweight     189
4  EM005      F   44    117  Overweight     183
```

绘制直方图命令如下：

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.hist(df['Age'],bins = 7)
plt.show()
```

最后得到如图 4-1 所示的结果。

通过观察直方图，可见各雇员年龄的分布情况。

上面的命令比较简单，分析过程及结果已经达到了解决实际问题的要求。但 R 的强大之处在于，它同样提供了更加复杂的命令格式，以满足用户更加个性化的需求。

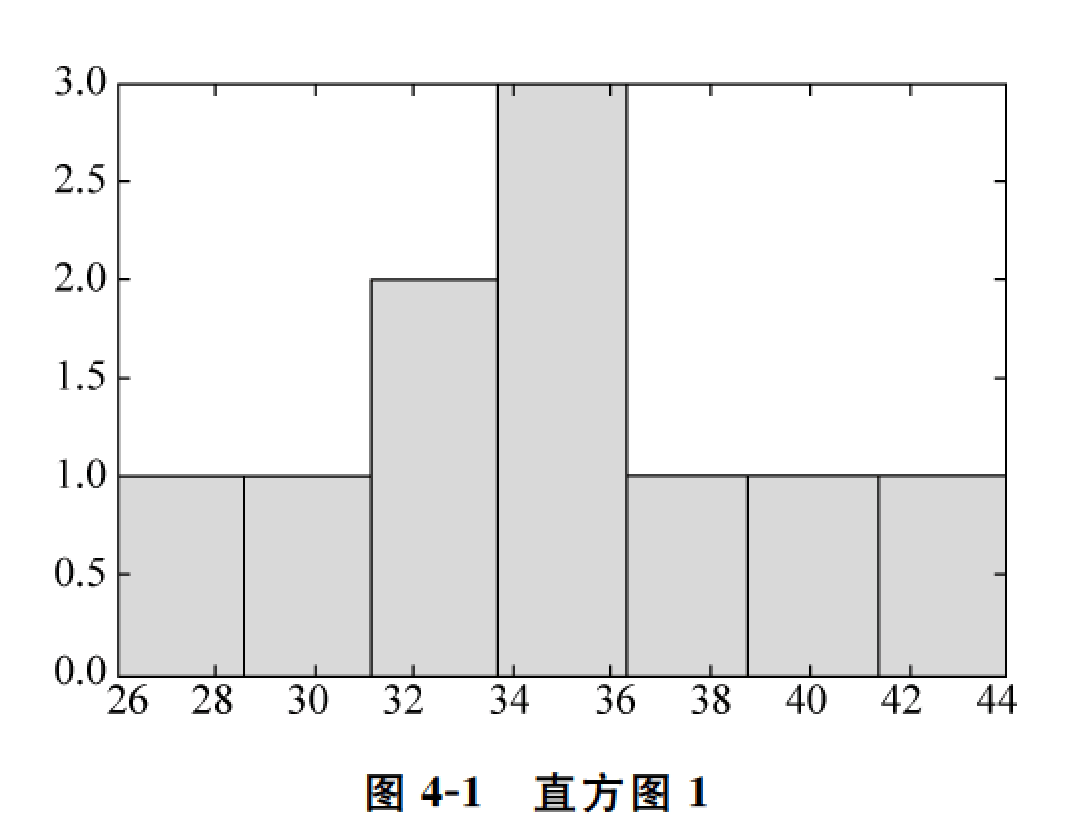


图 4-1　直方图 1

1. 给图形增加标题

例如，我们要给图形增加标题：Age distribution，那么上面的命令应为：

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.hist(df['Age'],bins = 7)
plt.title('Age distribution')
plt.show()
```

输入完成后，按回车键，得到如图 4-2 所示的结果。

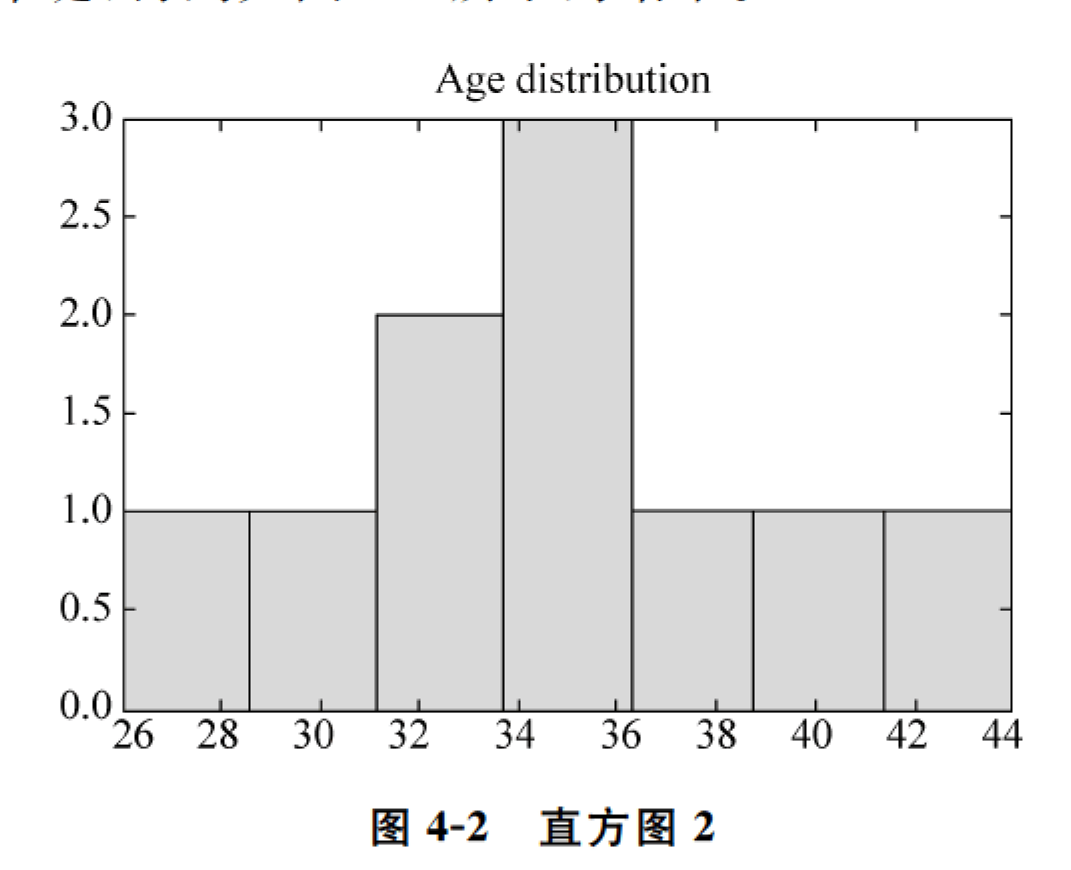


图 4-2　直方图 2

2. 给坐标轴增加数值标签

例如，我们要在图 4-2 的基础上对 X，Y 轴添加符号标签，那么上面的命令就应为：

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.hist(df['Age'],bins = 7)
plt.title('Age distribution')
plt.xlabel('Age')
plt.ylabel('#Employee')
plt.show()
```

输入完成后，按回车键，得到如图 4-3 所示的结果。

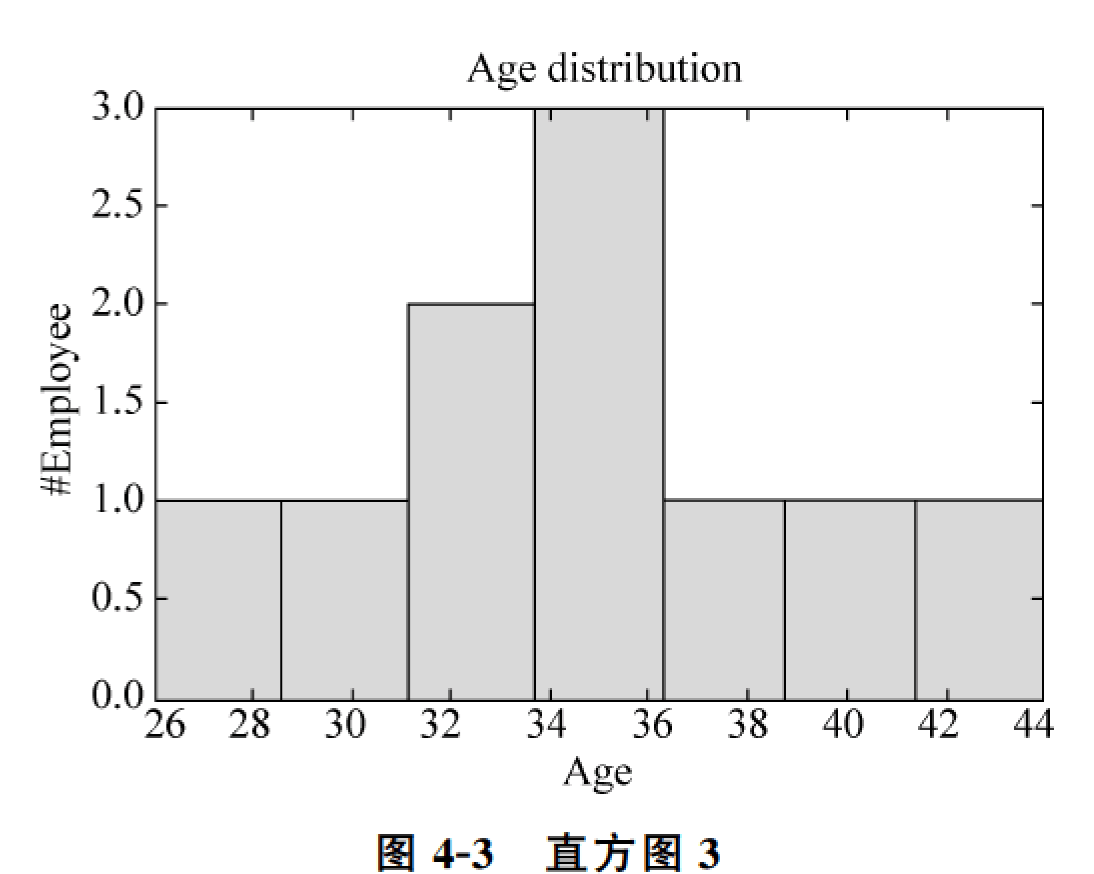


图 4-3　直方图 3

4.3　散点图的绘制

散点图就是点在坐标系平面上的分布图，它对数据预处理有很重要的作用。研究者对数据制作散点图的主要出发点是通过绘制该图来观察某变量随另一变量变化的大致趋势，据此可以探索数据之间的关联关系，甚至选择合适的函数对数据点进行拟合。

例 4-2　具体数据见例 4-1。

要绘制年龄、销售的散点图，输入如下命令：

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(df['Age'],df['Sales'])
#You can also add more variables here to represent color and size.
plt.title('Age & Sales Scatter of Employee')
#Variable
plt.xlabel('Age')
plt.ylabel('Sales')
plt.show()
```

输入完上述命令后，按回车键，得到如图 4-4 所示的结果。



图 4-4　散点图

通过观察图 4-4 所示的散点图，可以看出这些雇员的年龄和销售收入的有关情况。

4.4　气泡图的绘制

例 4-3　具体数据见例 4-1。

可以通过 scatter()中的 s 参数来绘制气泡图，如在绘制年龄与销售额的散点图时，通过气泡的大小来反映收入的大小。可以输入如下命令：

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(df['Age'],df['Sales'],s = df['Income'])
#Added third variable income as size of the bubble
plt.xlabel('Age')
plt.ylabel('Sales')
plt.show()
```

输入完上述命令后，按回车键，得到如图 4-5 所示的结果。

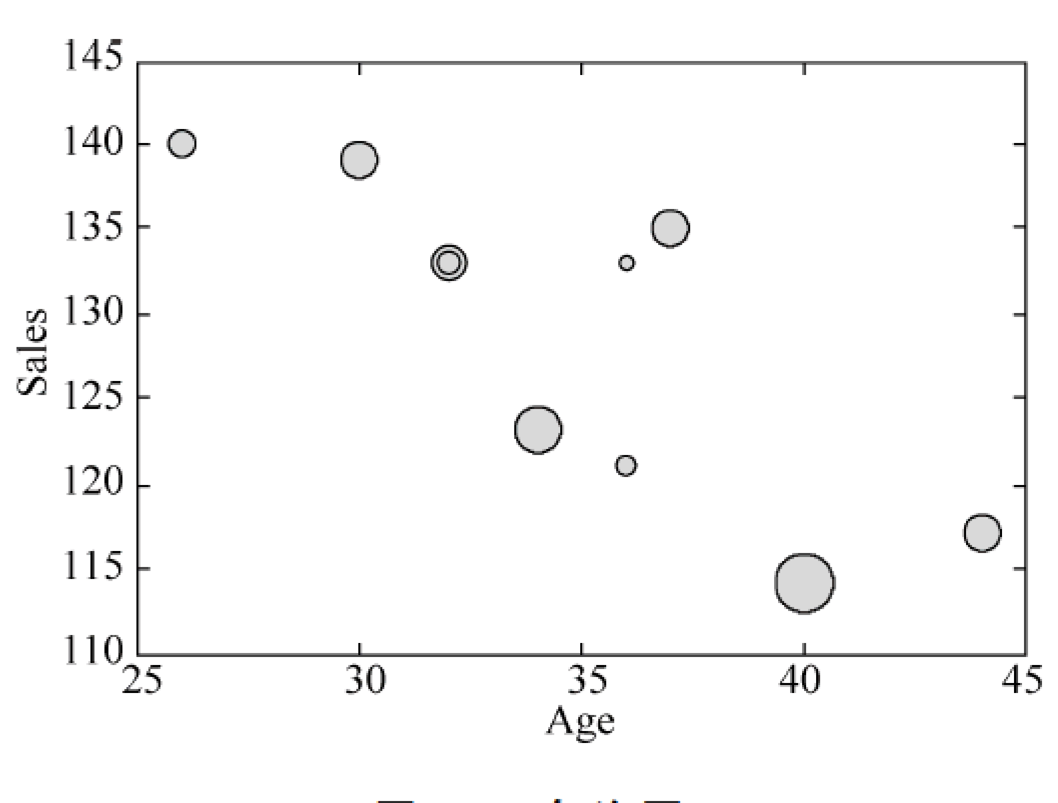


图 4-5　气泡图

通过观察图 4-5 所示的气泡图，可以看出这些雇员的年龄和销售收入的有关情况，还可以根据气泡的大小看出雇员的收入情况。

4.5　箱图的绘制

箱图又称为箱线图、盒须图，是一种用于显示一组数据分散情况的统计图。箱图很形象地分为中心、延伸以及分部状态的全部范围，它提供了一种只用 5 个点对数据集进行简单总结的方式，这 5 个点包括中点、Q1、Q3、分部状态的高位和低位。数据分析者通过绘制箱图可以直观明了地识别数据中的异常值，判断数据的偏态、尾重以及比较几批数据的形状。

例 4-4　具体数据见例 4-1。

要绘制年龄的箱图，输入如下命令：

```
import matplotlib.pyplot as plt
import Pandas as pd
fig = plt.figure()
```

```
ax = fig.add_subplot(1,1,1)
#Variable
ax.boxplot(df['Age'])
plt.title('Box figure of Age')
plt.show()
```

输入上述命令后,按回车键,得到如图 4-6 所示的结果。

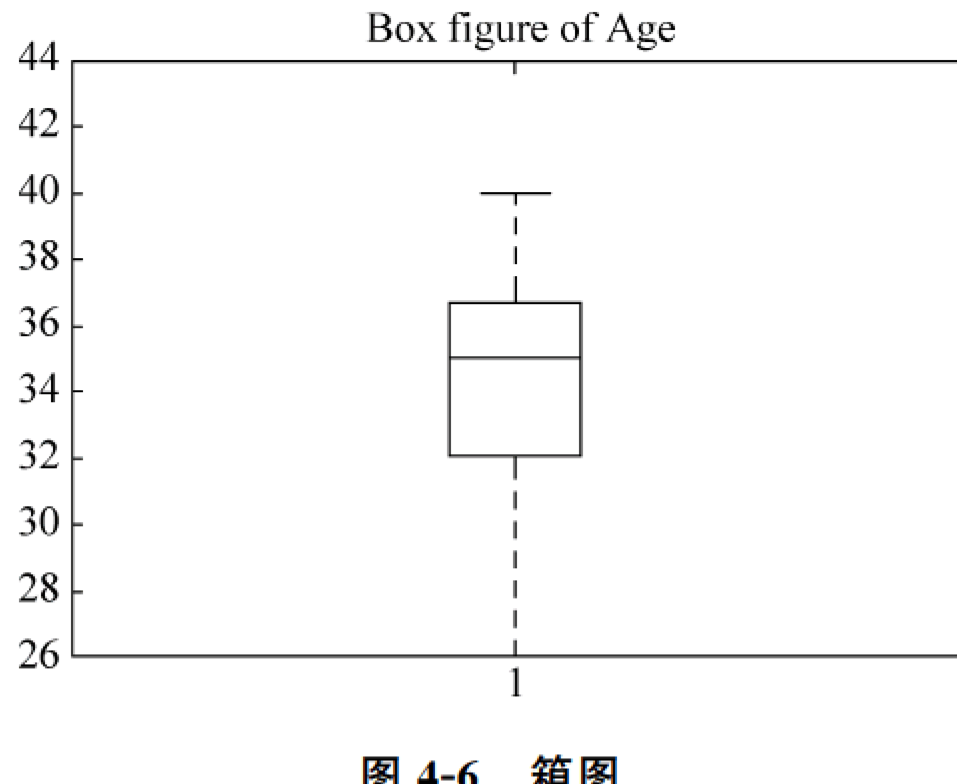


图 4-6　箱图

通过观察箱图 4-6,可以了解到很多信息。箱图把所有数据分成了 4 部分,第 1 部分是从顶线到箱子的上部,这部分数据值在全体数据中排名前 25%;第 2 部分是从箱子的上部到箱子中间的线,这部分数据值在全体数据中排名前 25%以下,50%以上;第 3 部分是从箱子的中间到箱子底部的底线,这部分数据值在全体数据中排名前 50%以下 75%以上;第 4 部分是从箱子的底部到底线,这部分数据值在全体数据中排名后 25%。顶线和底线的间距在一定程度上表示了数据的离散程度,间距越大就越离散。就本例而言,可以看到年龄的中位数在 35 岁左右,年龄最高值可达到 40 岁。

若要绘制多个属性的箱图,可使用如下代码:

```
vars = ['Age','Sales']
data = df[vars]
plt.show(data.plot(kind = 'box'))
```

输入上述命令后,按回车键,得到如图 4-7 所示的结果。

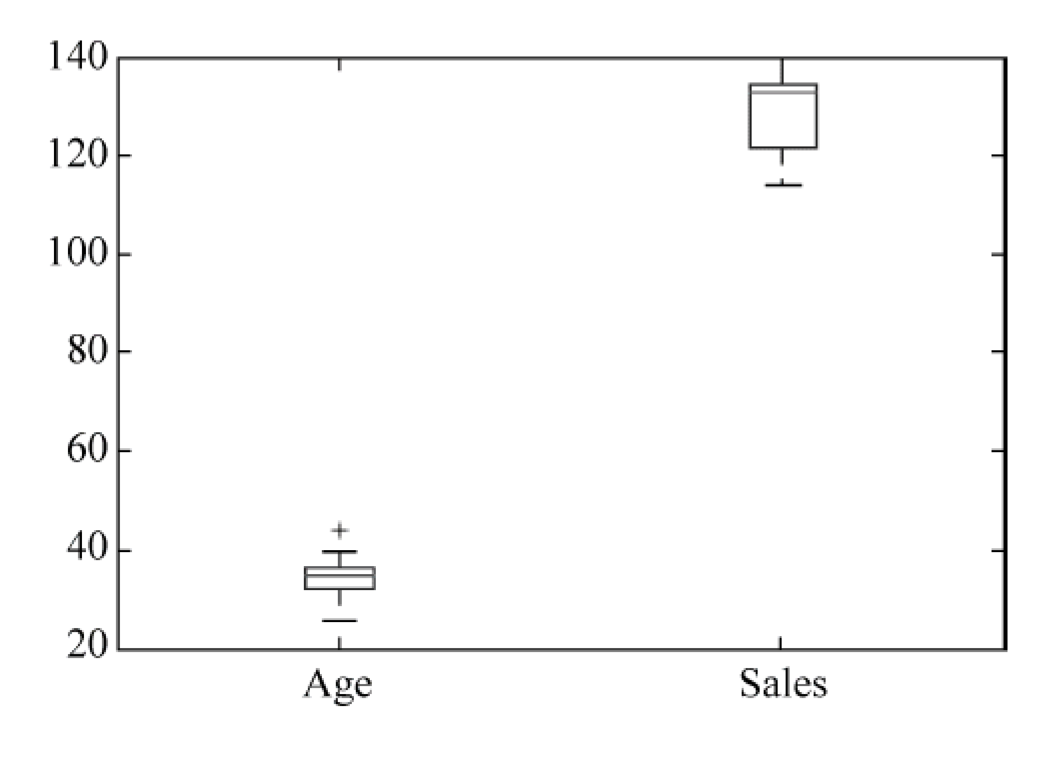


图 4-7　多属性箱图

4.6　饼图的绘制

饼图是数据分析中常见的一种经典图形，因其外形类似于圆饼而得名。在数据分析中，很多时候需要分析数据总体的各个组成部分的占比，我们可以通过各个部分与总额相除来计算，但这种数学比例的表示方法相对抽象，Python 提供了饼形制图工具，能够直接以图形的方式显示各个组成部分所占比例。更为重要的是，由于饼图采用图形的方式，因此更加形象直观。

4.6.1　简单饼图的绘制

例 4-5　具体数据见例 4-1。

在目录 G:\2glkx\data 下建立 al4-1.xls 数据文件后，使用如下命令读取数据：

```
import matplotlib.pyplot as plt
import Pandas as pd
import NumPy as np
df = pd.read_excel("G:/2glkx/data/al4 - 1.xls")
df.head()
```

得到显示前 5 条记录的数据如下：

```
   EMPID Gender  Age  Sales         BMI  Income
0  EM001      M   34    123      Normal     350
1  EM002      F   40    114  Overweight     450
2  EM003      F   37    135     Obesity     169
3  EM004      M   30    139  Overweight     189
4  EM005      F   44    117  Overweight     183
```

在 IPython console 输入如下命令：

```
var = df.groupby(['Gender']).sum().stack()
temp = var.unstack()
x_list = temp['Sales']
label_list = temp.index
plt.axis("equal")
plt.pie(x_list)
plt.title("Pastafatianism expenses")
plt.show()
```

输入完上述命令后，按回车键，得到如图 4-8 所示的结果。

通过观察该饼图(图 4-8)，可以看出该公司的雇员销售收入的情况，男雇员销售收入占 60%左右，女雇员销售收入占 40%左右。

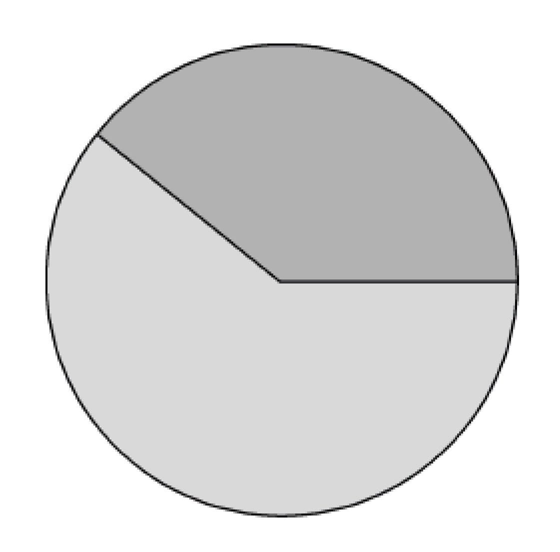


图 4-8　简单饼图

4.6.2　复杂饼图的绘制

下面给出一个绘制复杂饼图的程序。

```
from pylab import *
# make a square figure and axes
figure(1, figsize=(6,6))
ax = axes([0.1, 0.1, 0.8, 0.8])
fracs = [60, 40]                    #每一块占的比例,总和为 100
explode=(0, 0.08)                   #离开整体的距离,看效果
labels = 'Male', 'Female'           #对应每一块的标志
pie(fracs,explode=explode,labels=labels,autopct='%1.1f%%', shadow=True, startangle=
90, colors = ("g", "r"))
title('Rate of Male and Female')    #标题
show()
```

输入完上述命令后,按回车键,得到如图 4-9 所示的结果。

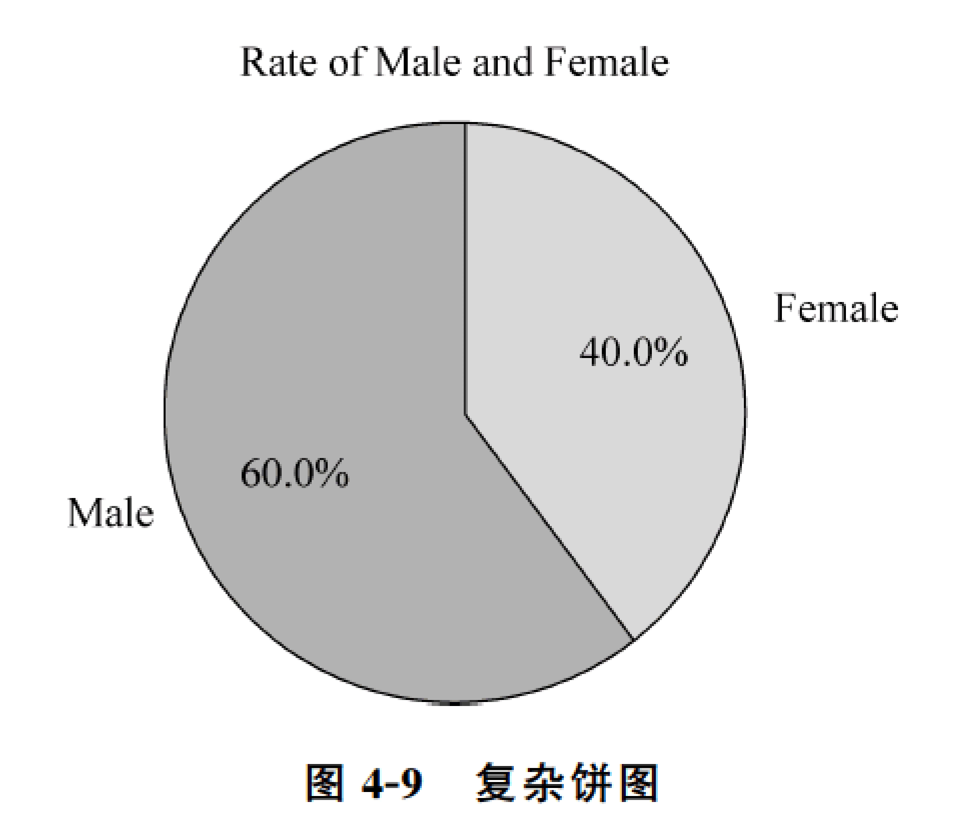


图 4-9　复杂饼图

4.7　条形图的绘制

相对于前面介绍的箱图,条形图(Bar Chart)本身所包含的信息相对较少,但是它仍然为平均数、中位数、合计数或计数等多种统计提供了简单而又多样化的展示,所以条形图也深受研究者的喜爱,经常出现在研究者的论文或者调查报告中。

4.7.1　简单条形图的绘制

例 4-6　具体数据见例 4-1。

在目录 G:\2glkx\data 下建立 al4-1. xls 数据文件后,使用如下命令读取数据:

```
import matplotlib.pyplot as plot
import Pandas as pd
import NumPy as np
df = pd.read_excel("G:/2glkx/data/al4-1.xls")
df.head()
```

得到显示前 5 条记录的数据如下:

```
   EMPID Gender  Age  Sales        BMI  Income
0  EM001      M   34    123     Normal     350
1  EM002      F   40    114 Overweight     450
2  EM003      F   37    135    Obesity     169
3  EM004      M   30    139 Overweight     189
4  EM005      F   44    117 Overweight     183
```

在 IPython console 输入如下命令，可以制作条形图。

```
var = df.groupby('Gender').Sales.sum()
#grouped sum of sales at Gender level
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.set_xlabel('Gender')
ax1.set_ylabel('Sum of Sales')
ax1.set_title("Gender wise Sum of Sales")
var.plot(kind = 'bar')
```

输入完上述命令后，按回车键，得到如图 4-10 所示的结果。

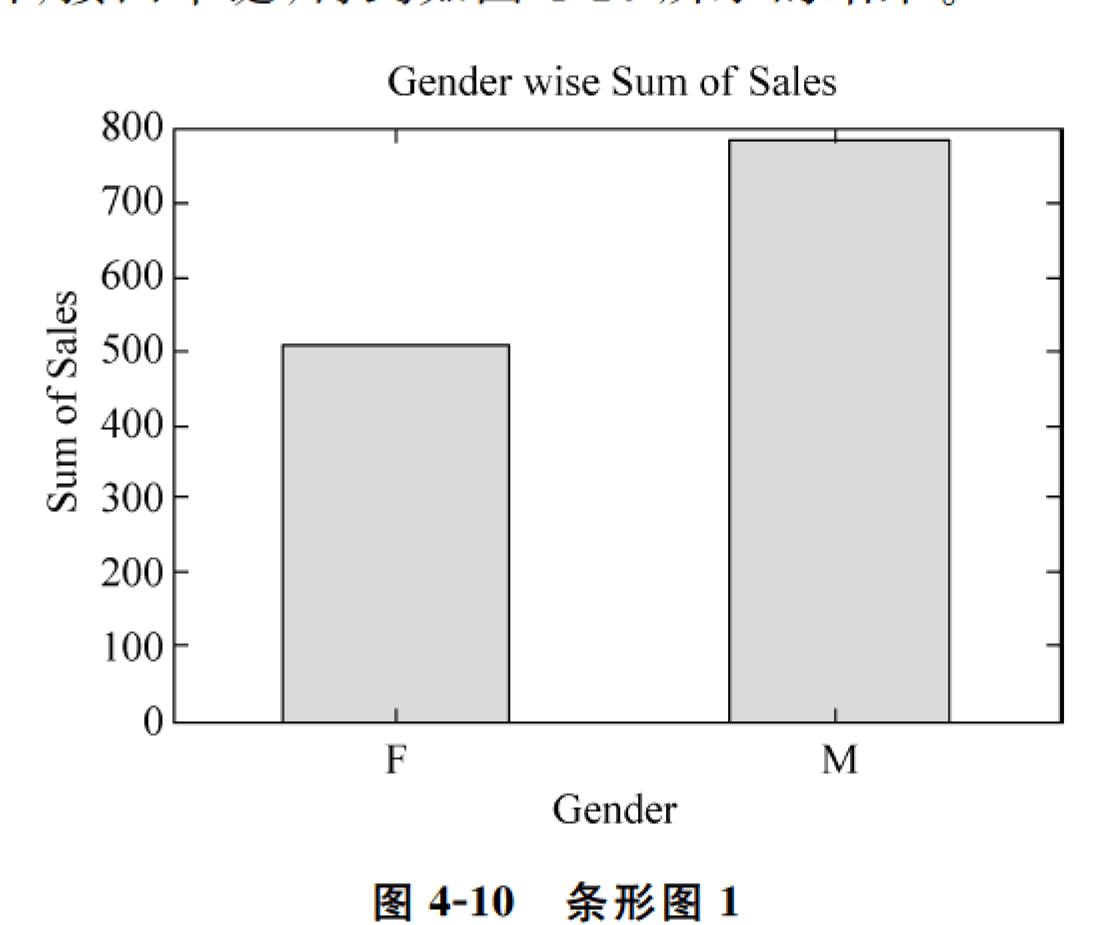


图 4-10 条形图 1

通过观察该条形图，可以看出该公司男性雇员的销售总额较高，女性雇员的销售总额较低。

4.7.2 复杂条形图的绘制

若我们先按体质指数 BMI 分类，在每一类体质指数 BMI 中按性别来展示总销售额，可以输入如下命令：

```
var = df.groupby(['BMI','Gender']).Sales.sum()
var.unstack().plot(kind = 'bar',stacked = True,color = ['red','blue'])
```

输入完上述命令后，按回车键，得到如图 4-11 所示的结果。

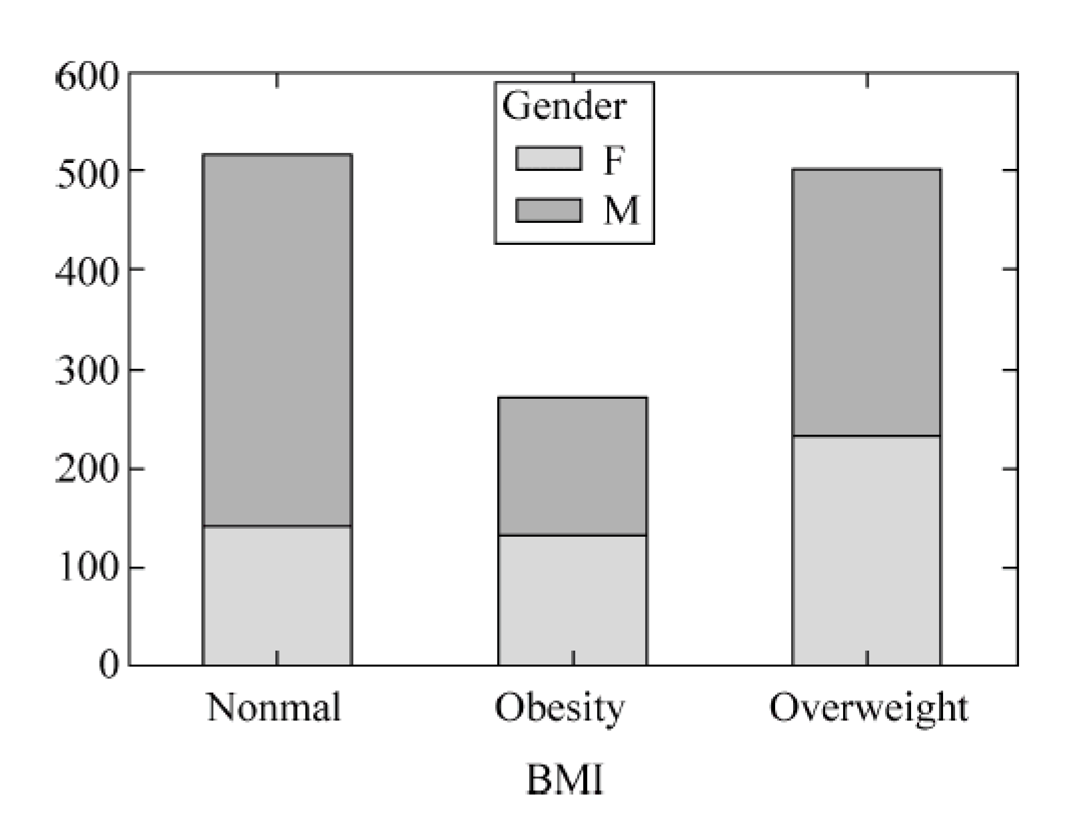


图 4-11 复杂条形图(或叫堆积柱形图)

4.8 折线图的绘制

例 4-7 具体数据见例 4-1。

在目录 G:\2glkx\data 下建立 al4-1. xls 数据文件后,使用如下命令读取数据:

```
import matplotlib.pyplot as plot
import Pandas as pd
import NumPy as np
df = pd.read_excel("G:/2glkx/data/al4 - 1.xls")
df.head()
```

得到显示前 5 条记录的数据如下:

	EMPID	Gender	Age	Sales	BMI	Income
0	EM001	M	34	123	Normal	350
1	EM002	F	40	114	Overweight	450
2	EM003	F	37	135	Obesity	169
3	EM004	M	30	139	Overweight	189
4	EM005	F	44	117	Overweight	183

在 IPython console 输入如下命令,可以制作曲线标绘图。

```
var = df.groupby('BMI').Sales.sum()
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.set_xlabel('BMI')
ax1.set_ylabel('Sum of Sales')
ax1.set_title("BMI wise Sum of Sales")
var.plot(kind = 'line')
```

输入完上述命令后,按回车键,得到如图 4-12 所示的结果。

从图 4-12 可以看出不同体质指数的销售总额情况。体质指数为正常或过重的雇员销售总额差不多,共 510 左右,体质指数为过胖的雇员销售总额较低,共 270 左右。

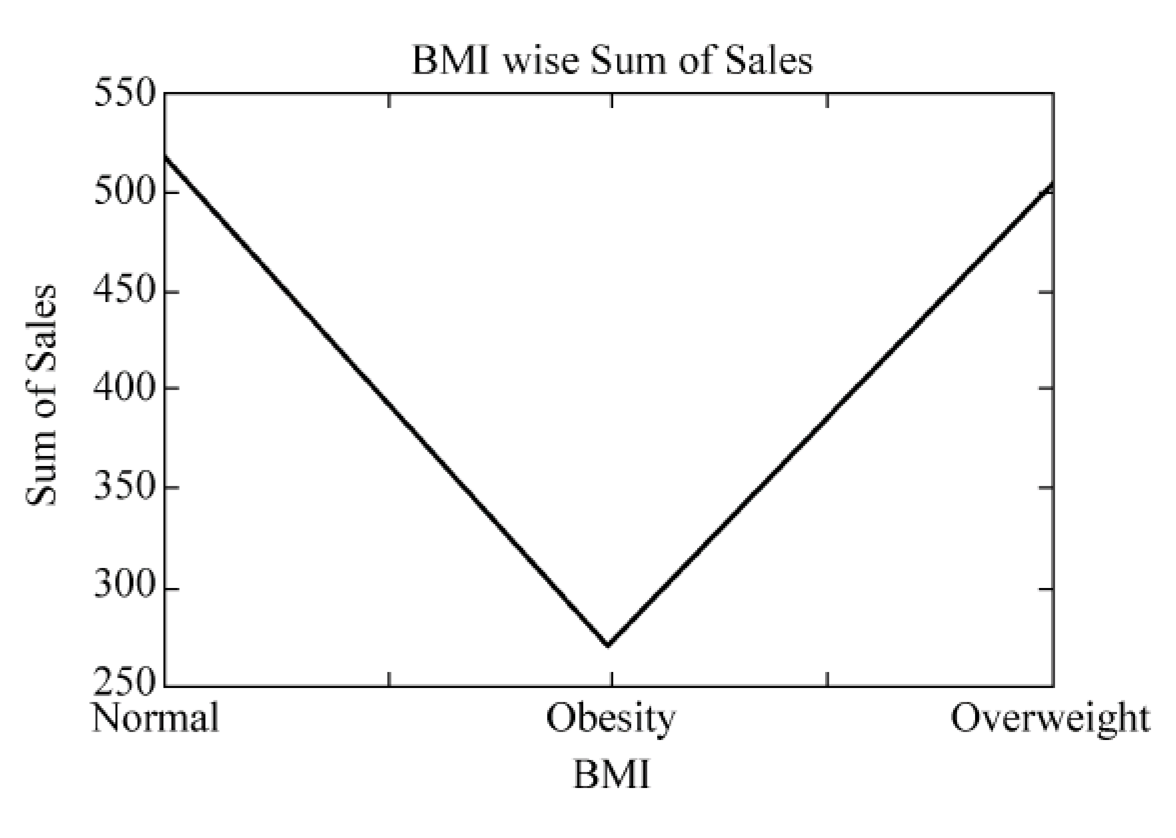


图 4-12　折线图

4.9　曲线标绘图的绘制

从形式上来看，曲线标绘图与散点图的区别就是以一条线替代散点标志，这样做可以更加清晰直观地看出数据走势，但却无法观察到每个散点的准确定位。从用途上看，曲线标绘图常用于时间序列分析的数据预处理，用来观察变量随时间的变化趋势。此外，曲线标绘图可以同时反映多个变量随时间的变化情况，所以，曲线标绘图的应用范围还是非常广泛的。

例 4-8　某村有每年自行进行人口普查的习惯，该村近年的人口数据如表 4-2 所示。试通过绘制曲线标绘图来分析研究该村的人口情况变化趋势以及新生儿对总人口数的影响程度。

表 4-2　某村人口普查资料

年份(year)	总人数(total)	新生儿数(new)
1997	128	15
1998	138	16
1999	144	16
2000	156	17
2001	166	21
2002	175	17
2003	180	18
2004	185	17
2005	189	30
2006	192	34
2007	198	37
2008	201	42
2009	205	41
2010	210	39
2011	215	38
2012	219	41

在目录 G:\2glkx\data 下建立 al4-3. xls 数据文件后，使用如下的命令读取数据。

```
import Pandas as pd
import NumPy as np
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al4 - 3.xls '))
data.head()
```

得到如下前 5 条记录的数据。

```
   year  total  new
0  1997    128   15
1  1998    138   16
2  1999    144   16
3  2000    156   17
4  2001    166   21
```

将上面的数据框对象的数据放入数据变量中。命令如下：

```
t = np.array(data[['year']])
x = np.array(data[['total']])
y = np.array(data[['new']])
```

再输入如下绘图命令：

```
import pylab as pl
pl.plot(t, x)
pl.plot(t, y)
pl.show()
```

输入完命令后，按回车键，得到如图 4-13 所示的结果。

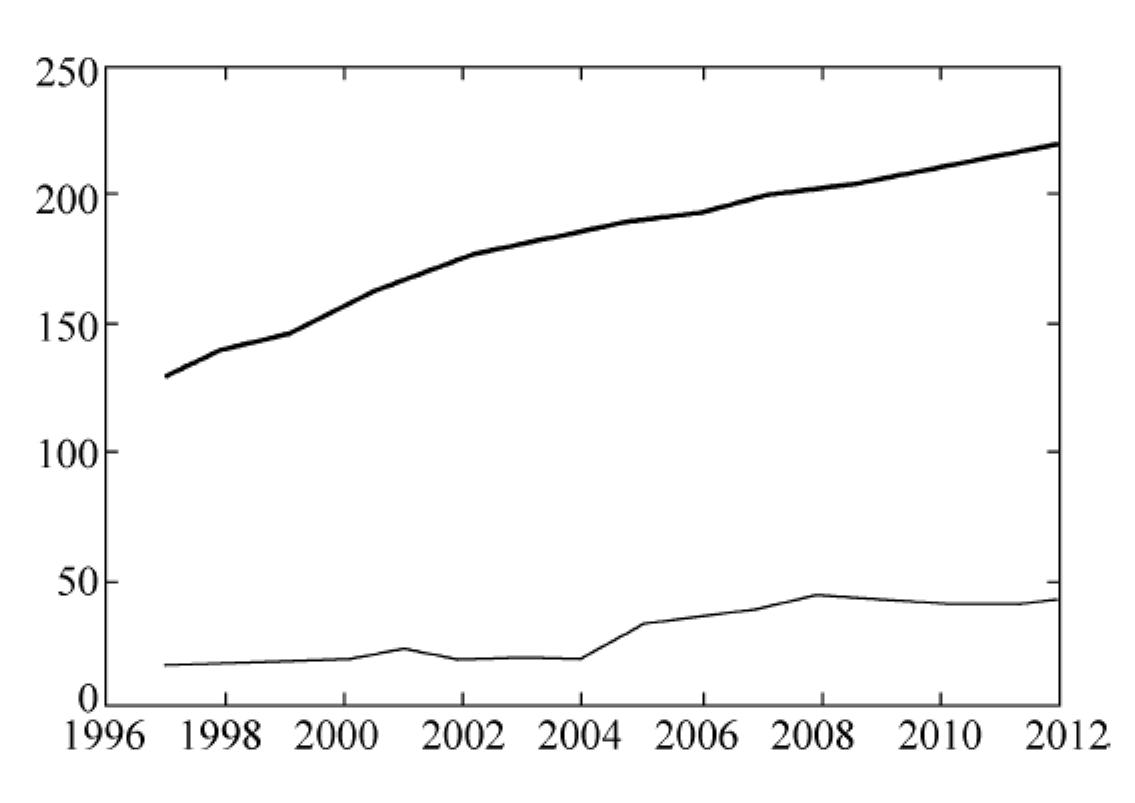


图 4-13 曲线标绘图 1

通过观察曲线图 4-13，可以看出该村总人数上升的速度快，新生儿小幅上升。

上面的 Python 命令比较简单，分析过程及结果已经达到解决实际问题的要求。但 Python 软件的强大之处在于，它同样提供了更加复杂的命令格式以满足用户更加个性化的需求。例如要给图形增加标题、给纵横坐标轴增加标签，则命令应为：

```
import pylab as pl
pl.plot(t, x)
pl.plot(t, y)
```

```
pl.title('Population census')
pl.xlabel('Time')
pl.ylabel('Population')
pl.show()
```

输入上述命令后,按回车键,得到如图 4-14 所示的结果。

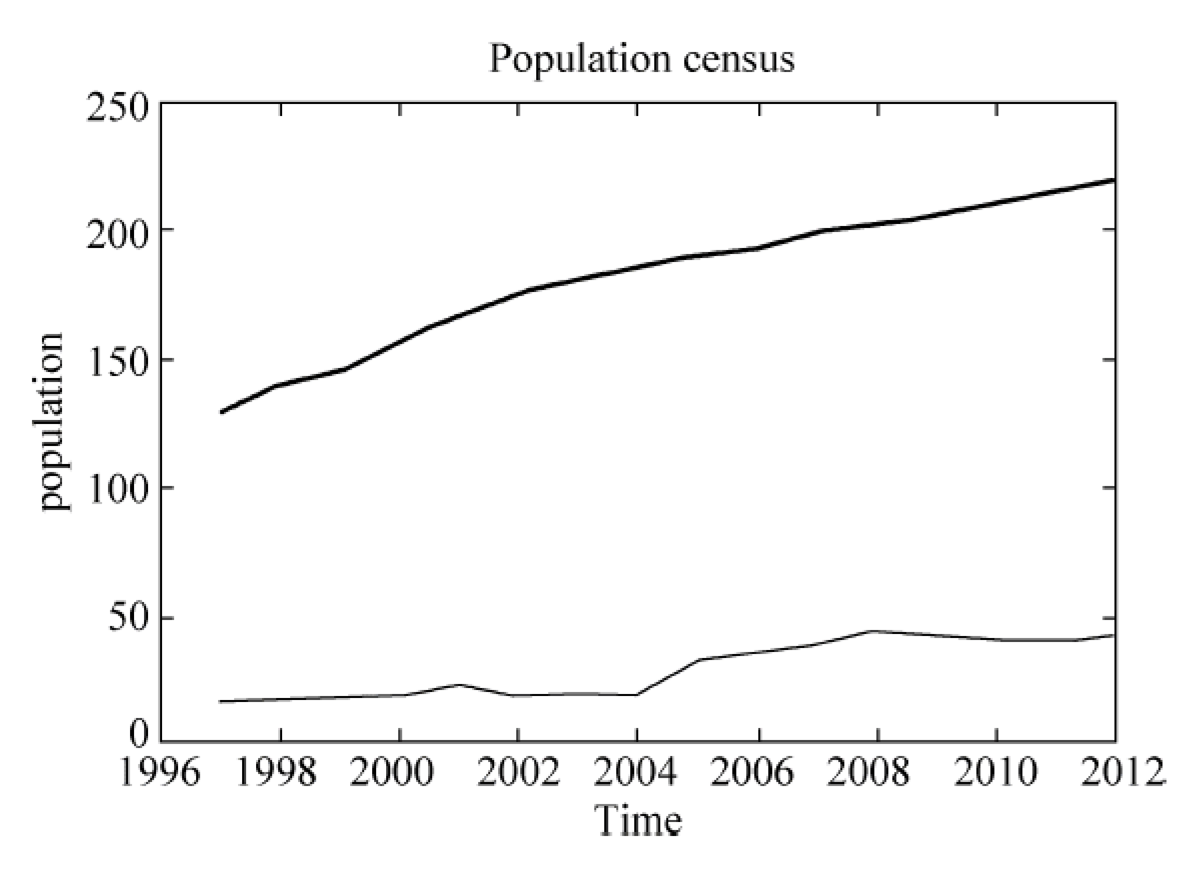


图 4-14　曲线标绘图 2

4.10　连线标绘图的绘制

在上节中我们提到的曲线标绘图用一条线来代替散点标志,可以更加清晰直观地看出数据走势,但却无法观察到每个散点的准确定位。如何做到既可以满足观测数据走势的需要,又能实现每个散点的准确定位呢? Python 的连线标绘图就可以解决这个问题。

例 4-9　1998—2015 年期间,我国上市公司的数量情况如表 4-3 所示。试通过绘制连线标绘图来分析研究我国上市公司数量的变化情况。

表 4-3　我国上市公司的数量情况

年　份	上市公司数量	年　份	上市公司数量
1998	851	2007	1550
1999	949	2008	1625
2000	1088	2009	1718
2001	1160	2010	2063
2002	1224	2011	2342
2003	1287	2012	2494
2004	1377	2013	2493
2005	1381	2014	2631
2006	1434	2015	2809

使用如下的命令读取数据。

```
import Pandas as pd
import NumPy as np
```

```
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al4 - 4.xls'))
data.head()
```

得到如下前 5 条记录的数据。

	year	number
0	1998	851
1	1999	949
2	2000	1088
3	2001	1160
4	2002	1224

将上面的数据框对象的数据放入数据变量中。命令如下：

```
t = np.array(data[['year']])
x = np.array(data[['number']])
```

再输入如下绘图命令：

```
import pylab as pl
pl.plot(t, x)
pl.title('1998 - 2015 of A listed companies in China')
pl.xlabel('Time')
pl.ylabel('Companies numbers')
pl.show()
```

输入完命令后，按回车键，得到如图 4-15 所示的结果。

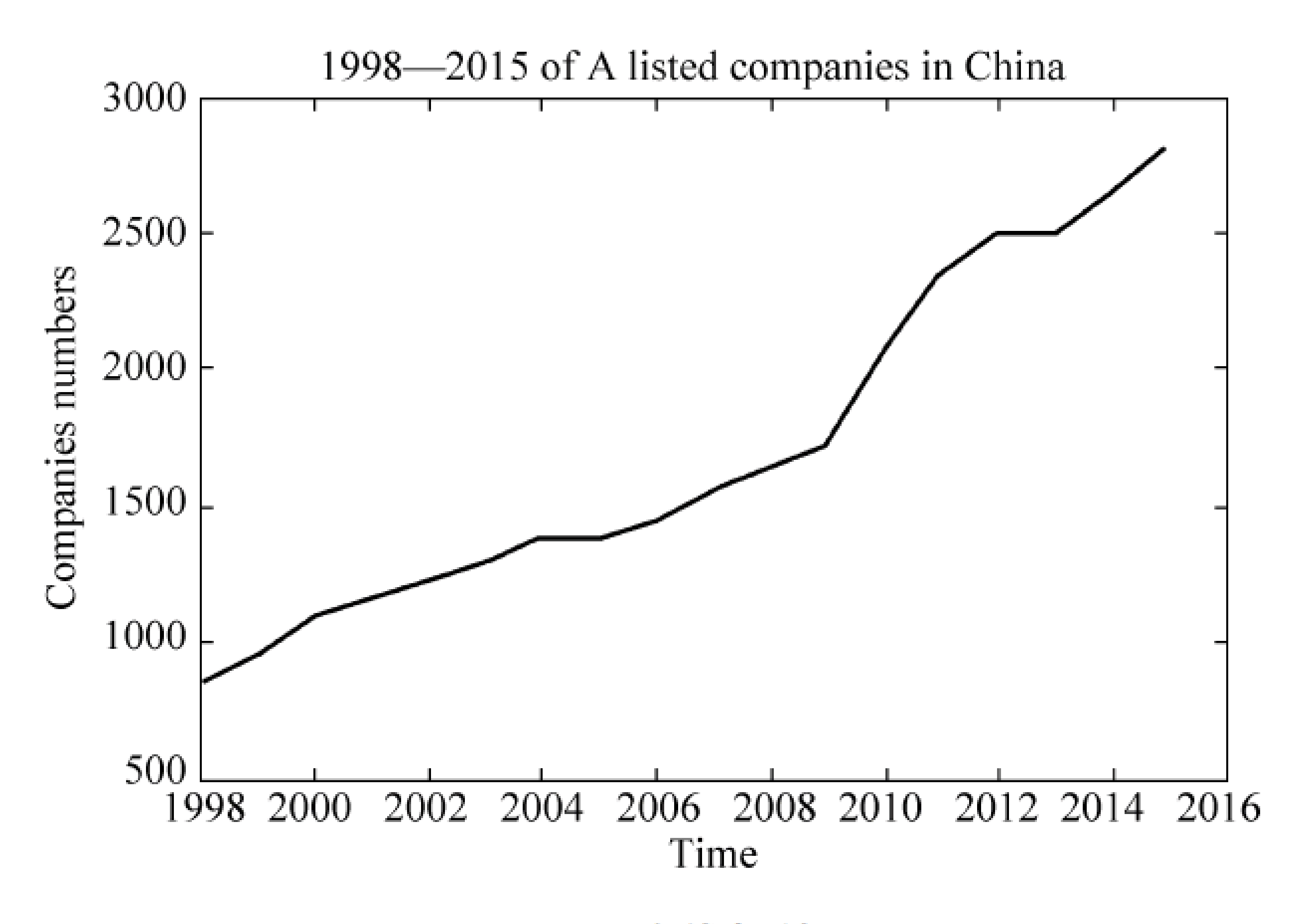


图 4-15 连线标绘图

通过观察连线标绘图 4-15，可以看出随着年份的增加，中国 A 股市场的上市公司数目基本上逐年增加，除了 2012 年到 2013 年有点小幅下降。

若要是上面的连线画成点，而非连线，则命令如下：

```
import Pandas as pd
import NumPy as np
import pylab as pl
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al4 - 4.xls'))
data.head()
```

```
t = np.array(data[['year']])
x= np.array(data[['number']])
import pylab as pl
pl.plot(t, x,'ro')
pl.title('1998 - 2015 of A listed companies in China')
pl.xlabel('Time')
pl.ylabel('Companies numbers')
pl.show()
```

输入完命令后,按回车键,得到如图 4-16 所示的结果。

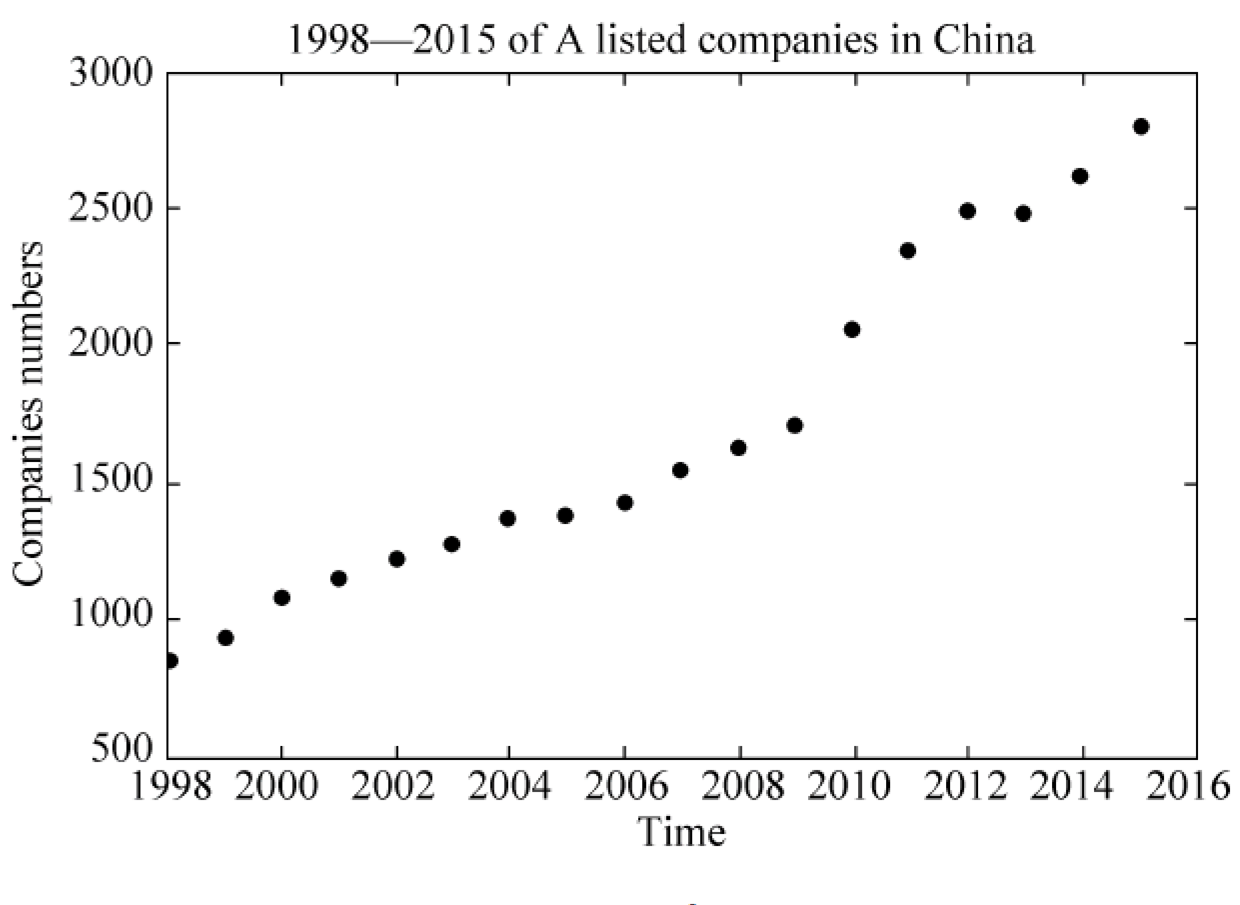


图 4-16 点图

通过观察图 4-16 的点图,也可以看出随着年份的增加,中国 A 股市场的上市公司数目基本上逐年增加,除了 2012 年到 2013 年有点小幅下降。

4.11 复杂图形的绘制

虽然 Matplotlib 主要专注于绘图,并且主要是二维的图形,但是它也有一些不同的扩展,能让我们在地理图上绘图,让我们把 Excel 和 3D 图表结合起来。在 Matplotlib 的世界里,这些扩展叫做工具包(toolkits)。比较流行的工具包有 Basemap、GTK 工具、Excel 工具、Natgrid、AxesGrid 和 Mplot3d。

本节探讨关于 mplot3d 的更多功能。mpl_toolkits. mplot3 工具包提供了一些基本的 3D 绘图功能,其支持的图表类型包括散点图(scatter)、曲面图(surf)、线图(line)和网格图(mesh)。虽然 mplot3d 不是一个最好的 3D 图形绘制库,但它是伴随着 Matplotlib 产生的。

我们现在需要创建一个图表并把想要的坐标轴添加到上面,但不同的是我们为图表指定的是 3D 视图,并且添加的坐标轴是 Axes3D。

现在,我们可以使用几乎相同的函数来绘图。当然,函数的参数是不同的,需要为 3 个坐标轴提供数据。例如,我们要为函数 mpl_toolkits. mplot3d. Axes3D. plot 指定 xs、ys、zs 和 zdir 参数。其他的参数则直接传给 matplotlib. axes. Axes. plot。

下面解释一下这些特定的参数:

(1) xs 和 ys: x 轴和 y 轴坐标;

(2) zs：这是 z 轴的坐标值，可以是所有点对应一个值，或者是每个点对应一个值；

(3) zdir：决定哪个坐标轴作为 z 轴的维度(通常是 zs，但是也可以是 xs 或者 ys)。

要注意的是：模块 mpl_toolkits. mplot3d. art3d 包含了 3D artist 代码和将 2D artists 转化为 3D 版本的函数。在该模块中有一个 rotate_axes 方法，该方法可以被添加到 Axes3D 中来对坐标重新排序，这样坐标轴就与 zdir 一起旋转了。zdir 默认值为 z。在坐标轴前加一个"－"会进行反转转换，这样一来，zdir 的值就可以是 x、－x、y、－y、z 或者－z。

以下代码展示了我们所解释的内容。

```
import random
import NumPy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from mpl_toolkits.mplot3d import Axes3D
mpl.rcParams['font.size'] = 10
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for z in [2011, 2012, 2013, 2014]:
     xs = xrange(1,13)
     ys = 1000 * np.random.rand(12)
     color = plt.cm.Set2(random.choice(xrange(plt.cm.Set2.N)))
     ax.bar(xs, ys, zs=z, zdir='y', color=color, alpha=0.8)
ax.xaxis.set_major_locator(mpl.ticker.FixedLocator(xs))
ax.yaxis.set_major_locator(mpl.ticker.FixedLocator(ys))
ax.set_xlabel('Month')
ax.set_ylabel('Year')
ax.set_zlabel('Sales Net [usd]')
plt.show()
```

输入完上述命令后，按回车键，得到如图 4-17 所示的结果。

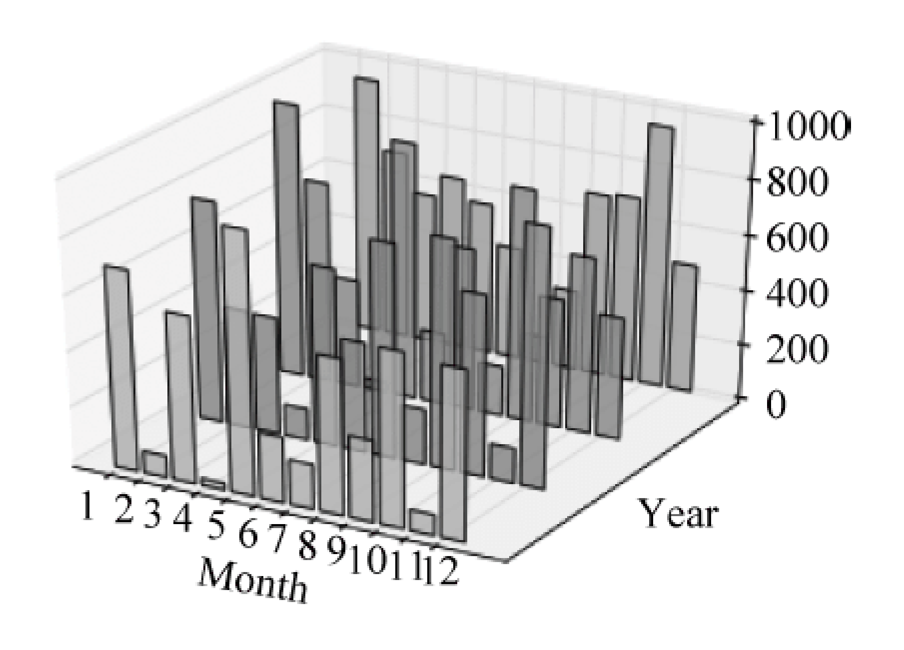


图 4-17 三维图

在下面的示例代码中，我们绘制了著名的 Pringle 函数的三翼面图，在数学上叫双曲面抛物线(hyperbolic paraboloid)。

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import NumPy as np
```

```
n_angles = 36
n_radii = 8
# An array of radii
# Does not include radius r = 0, this is to eliminate duplicate points
radii = np.linspace(0.125, 1.0, n_radii)
# An array of angles
angles = np.linspace(0, 2 * np.pi, n_angles, endpoint = False)
# Repeat all angles for each radius
angles = np.repeat(angles[..., np.newaxis], n_radii, axis = 1)
# Convert polar (radii, angles) coords to cartesian (x, y) coords
# (0,0)is added here.There are no duplicate points in the (x, y)plane
x = np.append(0, (radii * np.cos(angles)).flatten())
y = np.append(0, (radii * np.sin(angles)).flatten())
# Pringle surface
z = np.sin( - x * y)
fig = plt.figure()
ax = fig.gca(projection = '3d')
ax.plot_trisurf(x, y, z, cmap = cm.jet, linewidth = 0.2)
plt.show()
```

上面的代码生成如图 4-18 所示的图形。

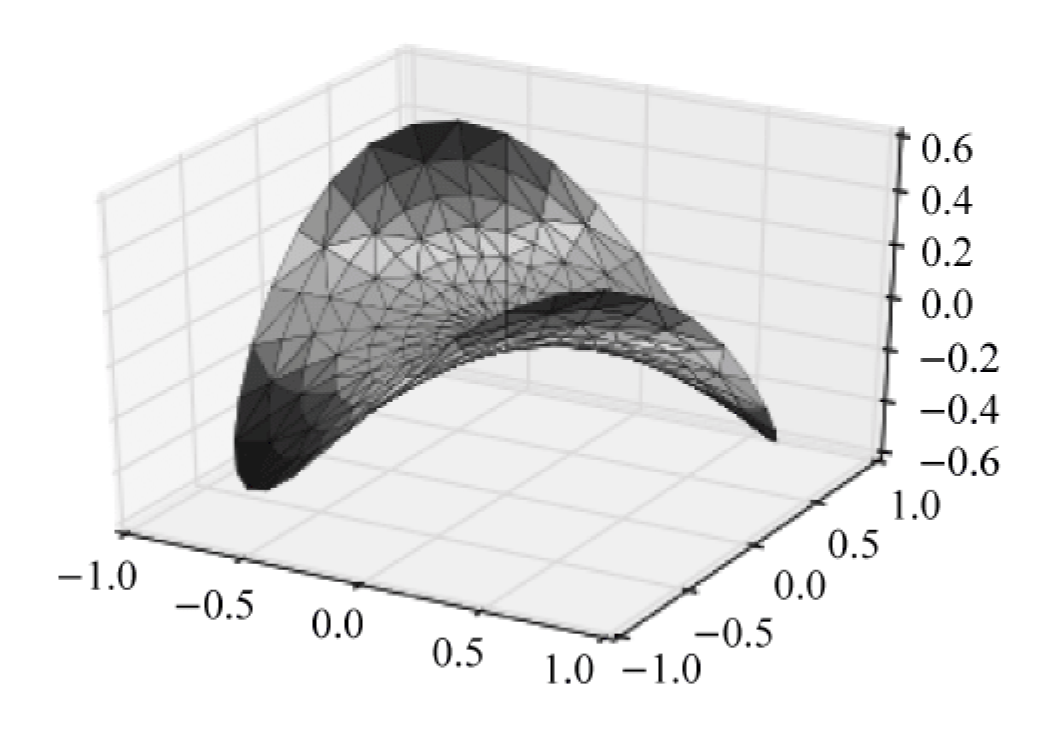


图 4-18 三翼面图(双曲面抛物线)

4.12 关于绘图中显示中文的问题处理

前面我们绘制的图形,都不能显示中文。因为 Matplotlib 的缺省配置文件中所使用的字体无法正确显示中文。为了在绘图中能正确显示中文,可以有几种解决方案。

(1) 在程序中直接指定字体;

(2) 在程序开头修改配置字典 rcParams;

(3) 修改配置文件。

下面代码实现了通过修改字体实现绘图中显示中文的问题。

```
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
import NumPy as np
font = FontProperties(fname = r"c:\windows\fonts\simsun.ttc", size = 14)
```

```
t = np.linspace(0, 10, 1000)
y = np.sin(t)
plt.plot(t, y)
plt.xlabel(u"时间", fontproperties = font)
plt.ylabel(u"振幅", fontproperties = font)
plt.title(u"正弦波", fontproperties = font)
plt.show()
```

上面的代码生成如图 4-19 所示的图形。

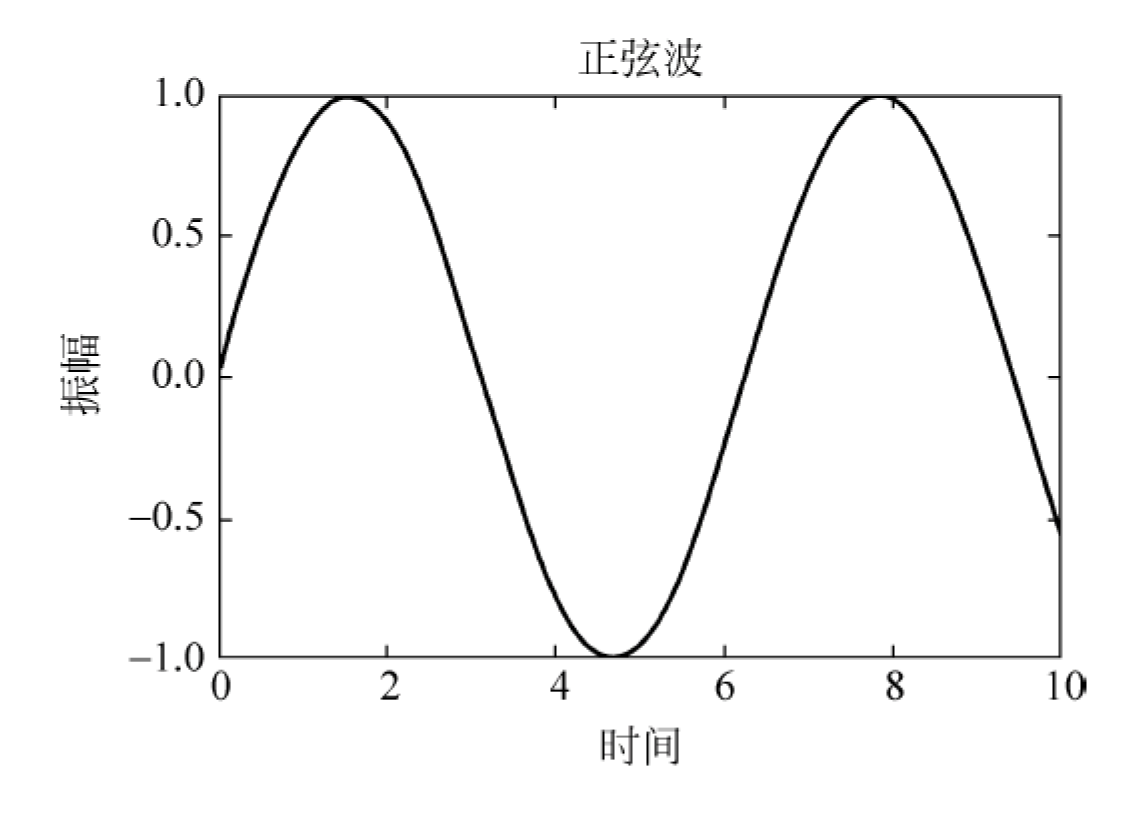


图 4-19 显示中文的图形

练 习 题

对本章例题中的数据文件,使用 Python 的绘图功能重新操作一遍。

第5章 概率统计分布的 Python 应用

在讨论概率统计分布之前，先说说什么是随机变量(random variable)。随机变量是对一次试验结果的量化。例如，一个表示抛硬币结果的随机变量可以表示成 $X=\{1$ 如果正面朝上，2 如果反面朝上$\}$。

随机变量是一个变量，它取值于一组可能的值(离散或连续的)，并服从某种随机性。随机变量的每个可能的取值都与一个概率相关。随机变量的所有可能取值和与之相关联的概率就被称为概率分布(probability distributrion)。

概率分布有两种类型：离散(discrete)概率分布和连续(continuous)概率分布。离散概率分布也称为概率质量函数(probability mass function)。离散概率分布的例子有伯努利分布(Bernoulli distribution)、二项分布(binomial distribution)、泊松分布(Poisson distribution)和几何分布(geometric distribution)等。连续概率分布也称为概率密度函数(probability density function)，它们是具有连续取值(例如一条实线上的值)的函数。正态分布(normal distribution)、指数分布(exponential distribution)和β分布(beta distribution)等都属于连续概率分布。若想了解更多关于离散和连续随机变量的知识，可以阅读概率分布的相关书籍。

5.1 二项分布

服从二项分布(binomial distribution)的随机变量 X 表示在 n 个独立的是/非试验中成功的次数，其中每次试验的成功概率为 p。

$$P(X=k)=\left(\frac{n!}{k!(n-k)!}\right)p^k(1-p)^{n-k}$$

$$E(X)=np,\quad \mathrm{var}(X)=np(1-p)$$

$E(X)$表示分布的期望或平均值，$\mathrm{var}(X)$表示分布的方差。

如果想知道每个函数的原理，可以在 IPython 笔记本中使用 help file 命令。键入 stats.binom，可以了解二项分布函数 binom 的更多信息。

例如：抛掷 10 次硬币，恰好两次正面朝上的概率是多少？

假设在该试验中正面朝上的概率为 0.3，这意味着平均来说，可以期待有 3 次是硬币正面朝上的。定义掷硬币的所有可能结果为 k=np.arange(0,11)：可能观测到 0 次正面朝上、1 次正面朝上，一直到 10 次正面朝上。使用 stats.binom.pmf 计算每次观测的概率质量函数。它返回一个含有 11 个元素的列表(list)，这些元素表示与每个观测相关联的概率值。

可以使用.rvs 函数模拟一个二项随机变量,其中参数 size 指定你要进行模拟的次数。让 Python 返回 10000 个参数为 n 和 p 的二项式随机变量。将输出这些随机变量的平均值和标准差,然后画出所有的随机变量的直方图。

```
import NumPy as np
from NumPy.random import randn
import SciPy
n = 10
p = 0.3
k = np.arange(0,21)
binomial = stats.binom.pmf(k,n,p)
binomial
```

得到如下结果:

```
array([  2.82475249e-02,   1.21060821e-01,   2.33474440e-01,
         2.66827932e-01,   2.00120949e-01,   1.02919345e-01,
         3.67569090e-02,   9.00169200e-03,   1.44670050e-03,
         1.37781000e-04,   5.90490000e-06,   0.00000000e+00,
         0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
         0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
         0.00000000e+00,   0.00000000e+00,   0.00000000e+00])
```

然后输入如下代码:

```
import matplotlib.pyplot as plt
plt.plot(k,binomial,'o-')
plt.title('binomial,n=%i,p=%.2f' % (n,p),fontsize=15)
plt.xlabel('Number of successes',fontsize=15)
plt.ylabel('Probability successes',fontsize=15)
plt.show()
```

执行上述代码后,得到如图 5-1 所示的结果。

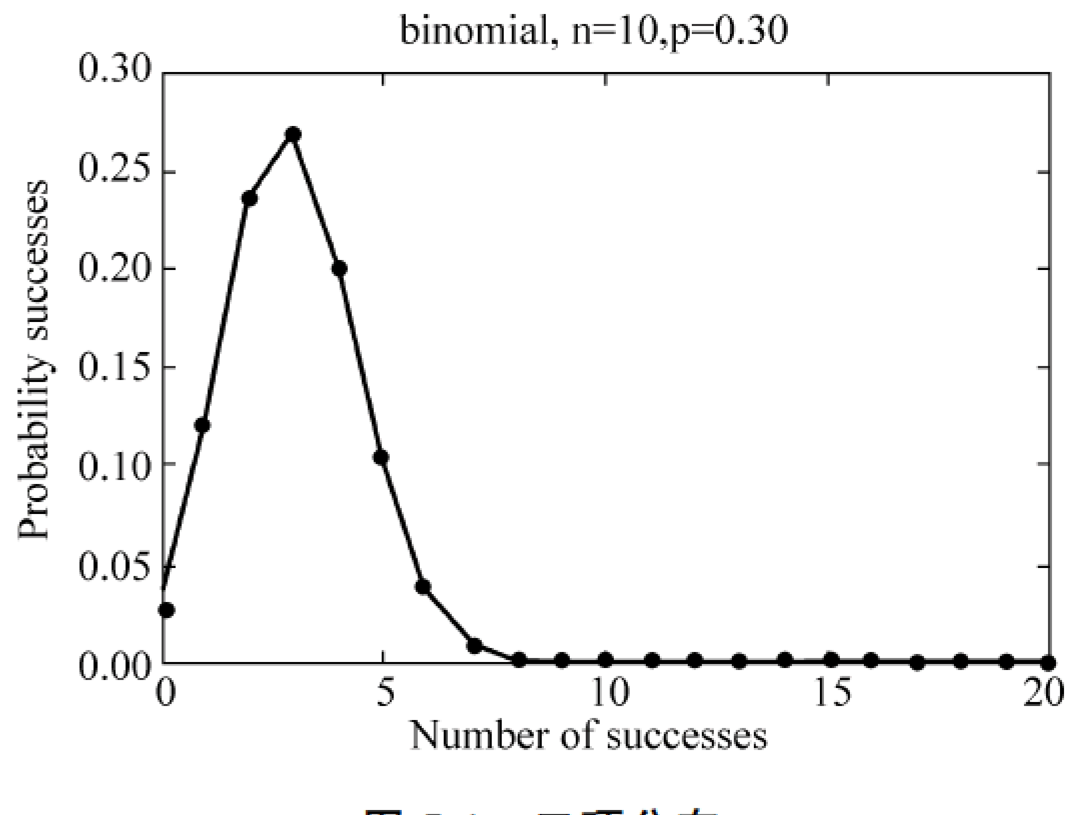


图 5-1　二项分布

可以使用.rvs 函数模拟一个二项随机变量,其中参数 size 指定要进行模拟的次数。让 Python 返回 10000 个参数为 n 和 p 的二项式随机变量。

Python 代码如下:

```
binom_sim = data = stats.binom.rvs(n = 10,p = 0.3,size = 10000)
print "Mean: %g" % np.mean(binom_sim)
print "SD: %g" % np.std(binom_sim,ddof = 1)
plt.hist(binom_sim,bins = 10,normed = True)
plt.xlabel("x")
plt.ylabel("density")
plt.show()
```

执行上述代码后，将输出这些随机变量的平均值和标准差，然后画出所有的随机变量的直方图，如图 5-2 所示。

```
Mean:3.0234
SD:1.44564
```

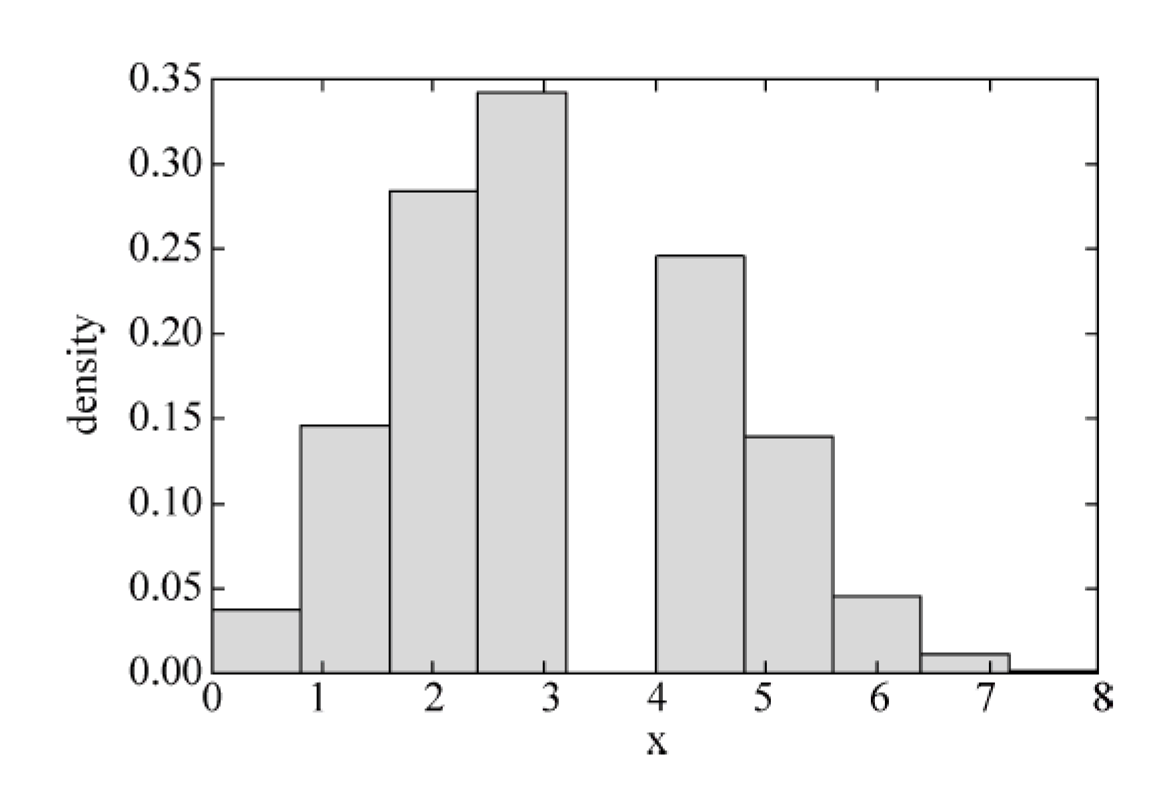


图 5-2　二项分布直方图

5.2　泊松分布

一个服从泊松分布（Poisson distribution）的随机变量 X，表示在具有比率参数（rate parameter）λ 的一段固定时间间隔内，事件发生的次数。参数 λ 告诉你该事件发生的比率。随机变量 X 的平均值和方差都是 λ。

$$P(X=k)=\frac{\lambda^k e^{-\lambda}}{k!},\quad E(X)=\lambda,\quad \mathrm{var}(X)=\lambda$$

泊松分布的实例：已知某路口发生事故的比率是每天 2 次，那么在此处一天内发生 4 次事故的概率是多少？

考虑这个平均每天发生 2 起事故的实例。泊松分布的实现和二项分布有些类似，在泊松分布中需要指定比率参数。泊松分布的输出是一个数列，包含了发生 0 次、1 次、2 次，直到 10 次事故的概率。

Python 代码如下：

```
rate = 2
n = np.arange(0,10)
y = stats.poisson.pmf(n,rate)
y
```

得到如下结果：

```
array([  1.35335283e-01,   2.70670566e-01,   2.70670566e-01,
         1.80447044e-01,   9.02235222e-02,   3.60894089e-02,
         1.20298030e-02,   3.43708656e-03,   8.59271640e-04,
         1.90949253e-04])
import matplotlib.pyplot as plt
plt.plot(y,'o-')
plt.show()
```

执行上述代码，得到如图 5-3 所示的结果。

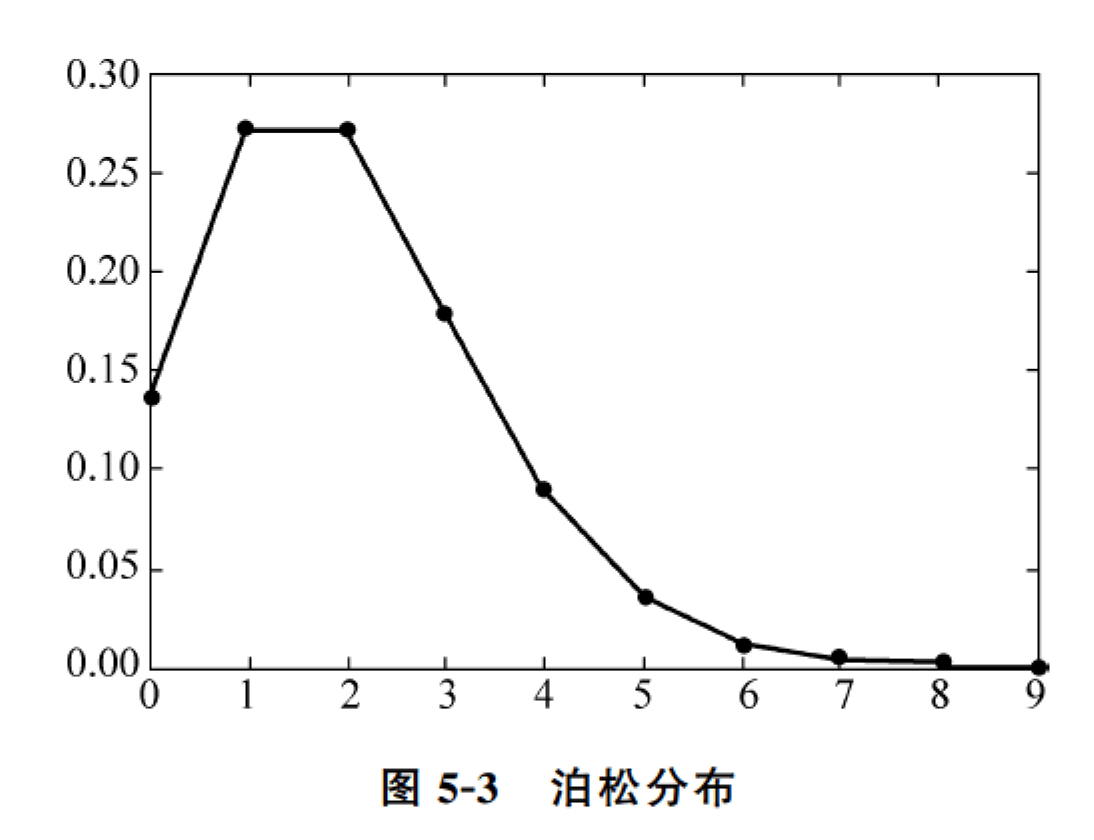


图 5-3　泊松分布

从图 5-3 可以看到，事故次数的峰值在均值附近。平均来说，可以预计事件发生的次数为 λ。尝试不同的 λ 和 n 的值，然后看看分布的形状是怎么变化的。

现在我来模拟 1000 个服从泊松分布的随机变量。Python 代码如下：

```
data = stats.poisson.rvs(mu = 2,loc = 0,size = 1000)
print "Mean: %g" % np.mean(data)
print "SD: %g" % np.std(data,ddof = 1)
plt.hist(data,bins = 10,normed = True)
plt.xlabel("numbers of accidents")
plt.ylabel("simulating poisson random variable")
plt.show()
```

执行上述代码后，将输出这些随机变量的平均值和标准差，然后画出所有的随机变量的直方图，如图 5-4 所示。

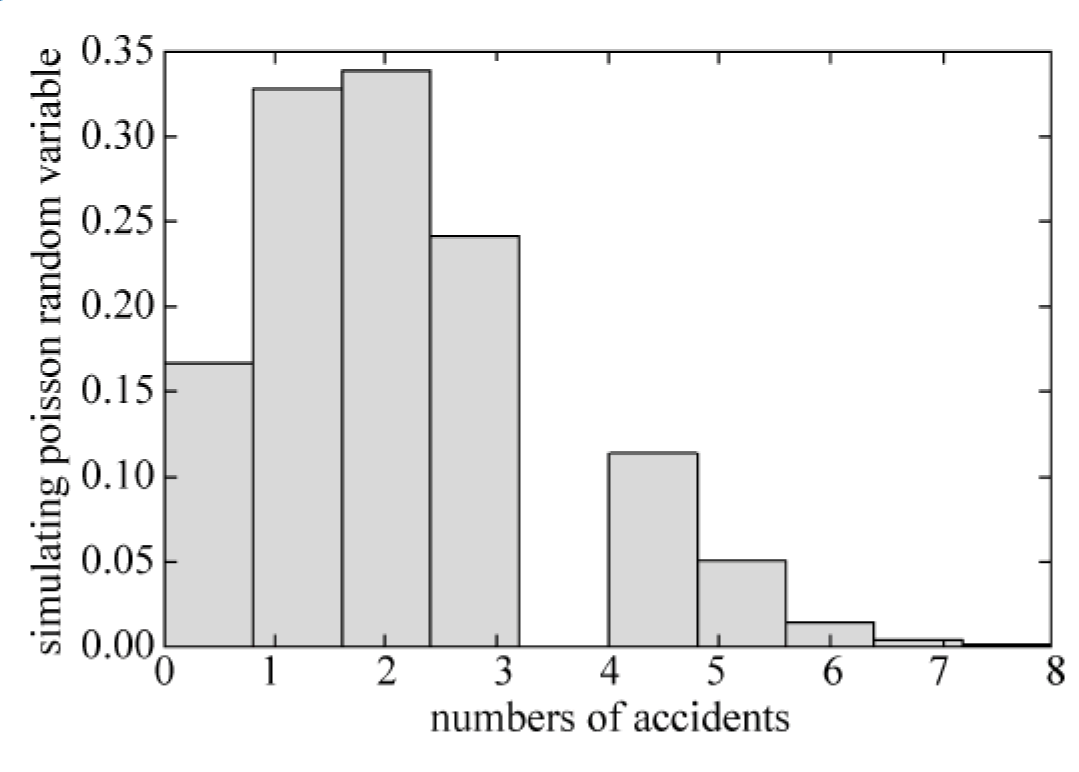


图 5-4　泊松分布直方图

```
Mean:2.02
SD:1.39339
```

5.3　正 态 分 布

正态分布(normal distribution)是一种连续分布,其函数可以在实线上的任何地方取值。正态分布由两个参数描述:分布的平均值 μ 和方差 σ^2。

$$P(x;\ \mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),\quad x \in (-\infty,\infty)$$

$$E(X) = \mu,\quad \mathrm{var}(X) = \sigma^2$$

正态分布的取值可以从负无穷到正无穷。可以用 stats.norm.pdf 得到正态分布的概率密度函数。正态分布的 Python 代码如下:

```
mu = 0
sigma = 1
x = np.arange( - 5,5,0.1)
y = stats.norm.pdf(x,0,1)
plt.plot(x,y)
plt.title('nomal, $ \mu $ = % .1f, $ sigma ^2 $ = % .1f' % (mu,sigma),fontsize = 15)
plt.xlabel("x")
plt.ylabel("Probability density")
plt.show()
```

执行上述代码,得到如图 5-5 所示的结果。

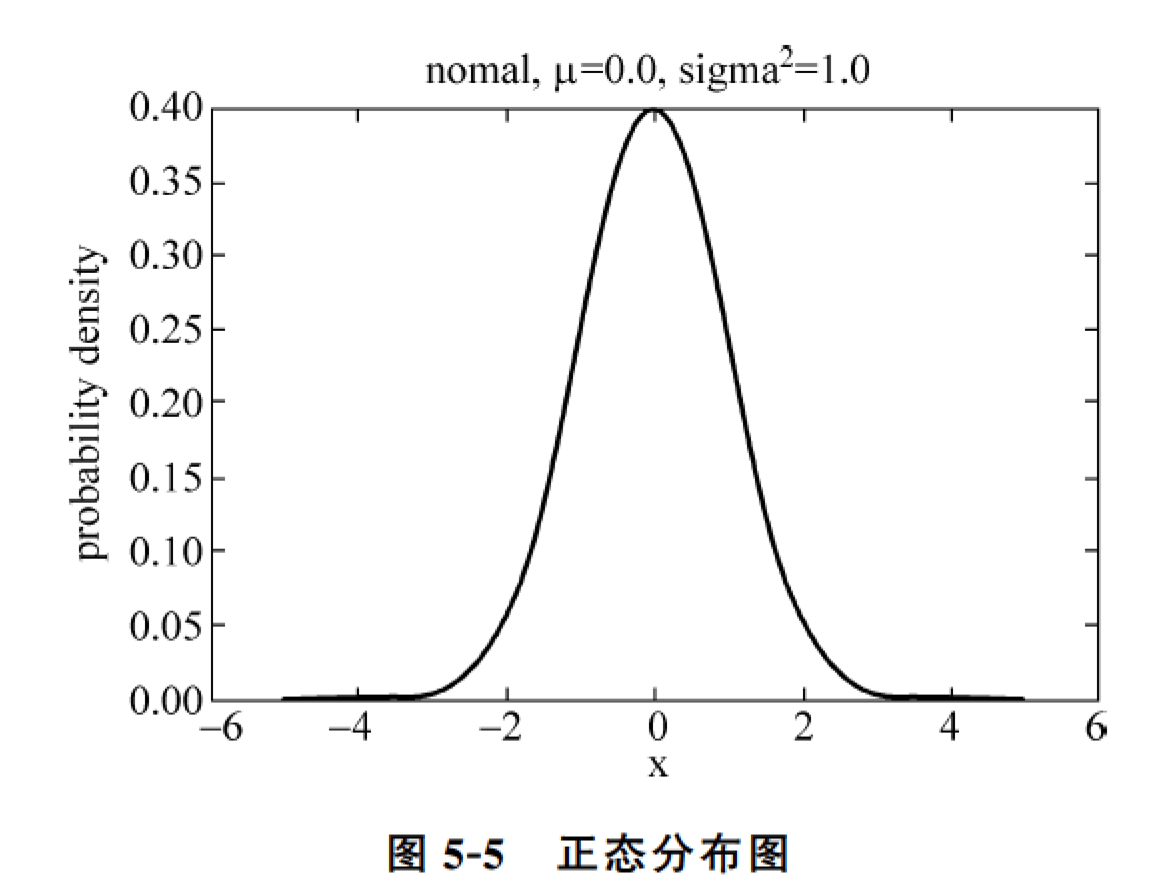


图 5-5　正态分布图

现在来模拟 1000 个均值为 0.0,标准差为 1.0 的服从正态分布的随机变量。Python 代码如下:

```
data =  stats.norm.rvs(loc = 0.0, scale = 1.0, size = 1000)
print "Mean: % g"  % np.mean(data)
print "SD: % g"  % np.std(data,ddof = 1)
plt.hist(data,bins = 20,normed = True)
plt.xlabel("numbers of accidents")
```

```
plt.ylabel("Norm distribution")
plt.show()
```

执行上述代码后,将输出这些随机变量的平均值和标准差,然后画出所有的随机变量的直方图,如图 5-6 所示。

```
Mean:-0.00227837
SD:1.04123
```

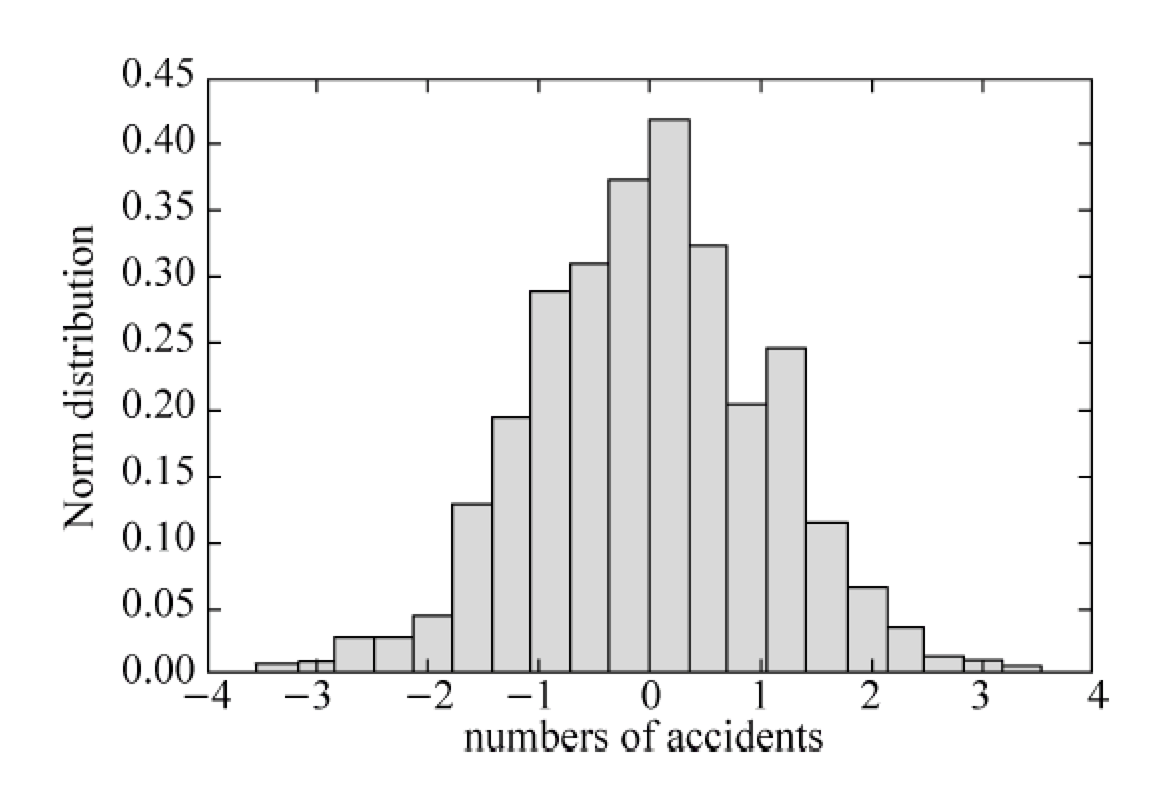


图 5-6　正态分布直方图

5.4　β 分布

β 分布(beta distribution)是一个取值在[0, 1]之间的连续分布,它由两个形态参数 α 和 β 的取值所描述。

$$P(x,\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}, \quad x \in [0,1], \alpha > 0, \beta > 0$$

$$E(x) = \frac{\alpha}{\alpha+\beta}, \quad \mathrm{var}(x) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

β 分布的形状取决于 α 和 β 的值。贝叶斯分析中大量使用了 β 分布。

β 分布的直方图 Python 代码如下:

```
a = 0.5
b = 0.5
x = np.arange(0.01,1,0.01)
y = stats.beta.pdf(x,a,b)
plt.plot(x,y)
print 'Beta:a = %.1f,b = %.1f'%,(a,b)
plt.xlabel("x")
plt.ylabel("Probability density")
plt.show()
```

执行上述代码,得到如图 5-7 所示的结果。

尝试不同的 α 和 β 取值,看看分布的形状是如何变化的。

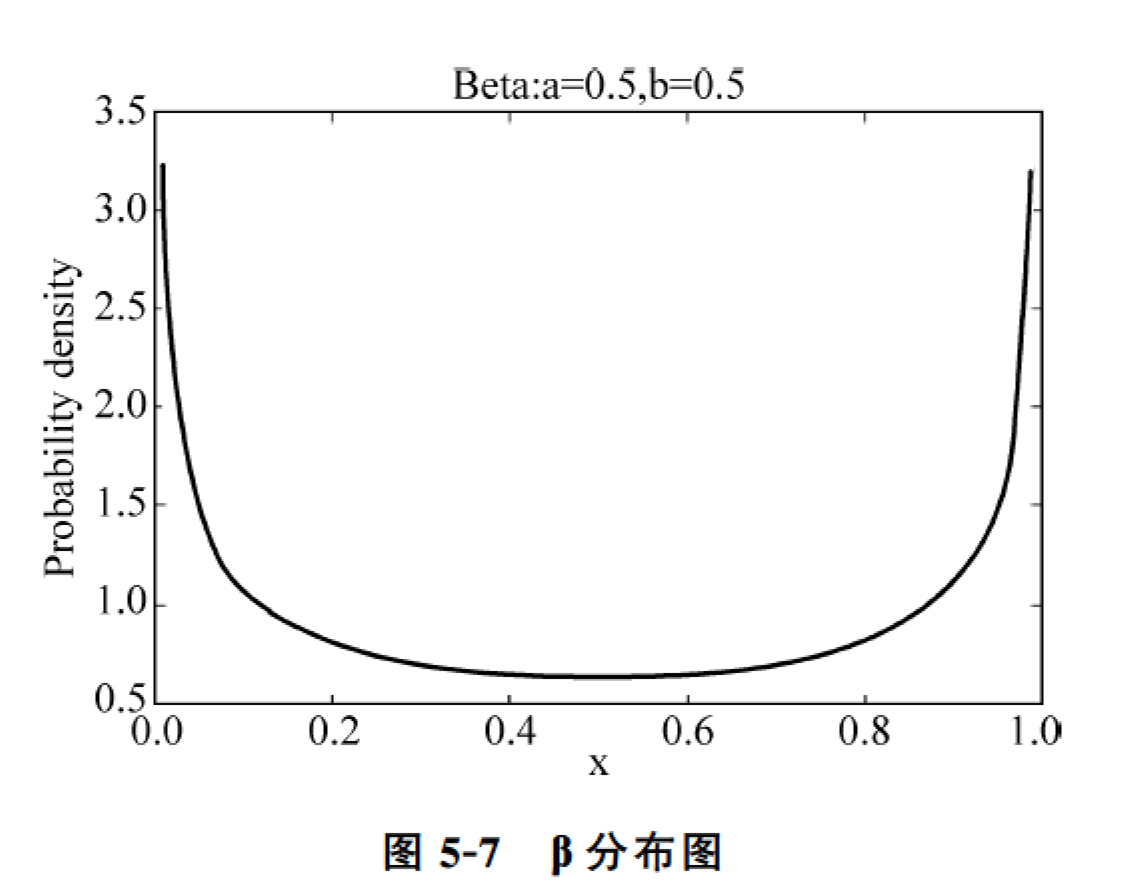


图 5-7　β 分布图

5.5　均匀分布

将参数 α 和 β 都设置为 1 时，该分布又被称为均匀分布(uniform distribution)。Python 代码如下：

```
a = 1.0
b = 1.0
x = np.arange(0.01,1,0.01)
y = stats.beta.pdf(x,a,b)
plt.plot(x,y)
plt.title('Beta:a = %.1f,b = %.1f' % (a,b))
plt.xlabel("x")
plt.ylabel("Probability density")
plt.show()
```

执行上述代码，得到如图 5-8 所示的结果。

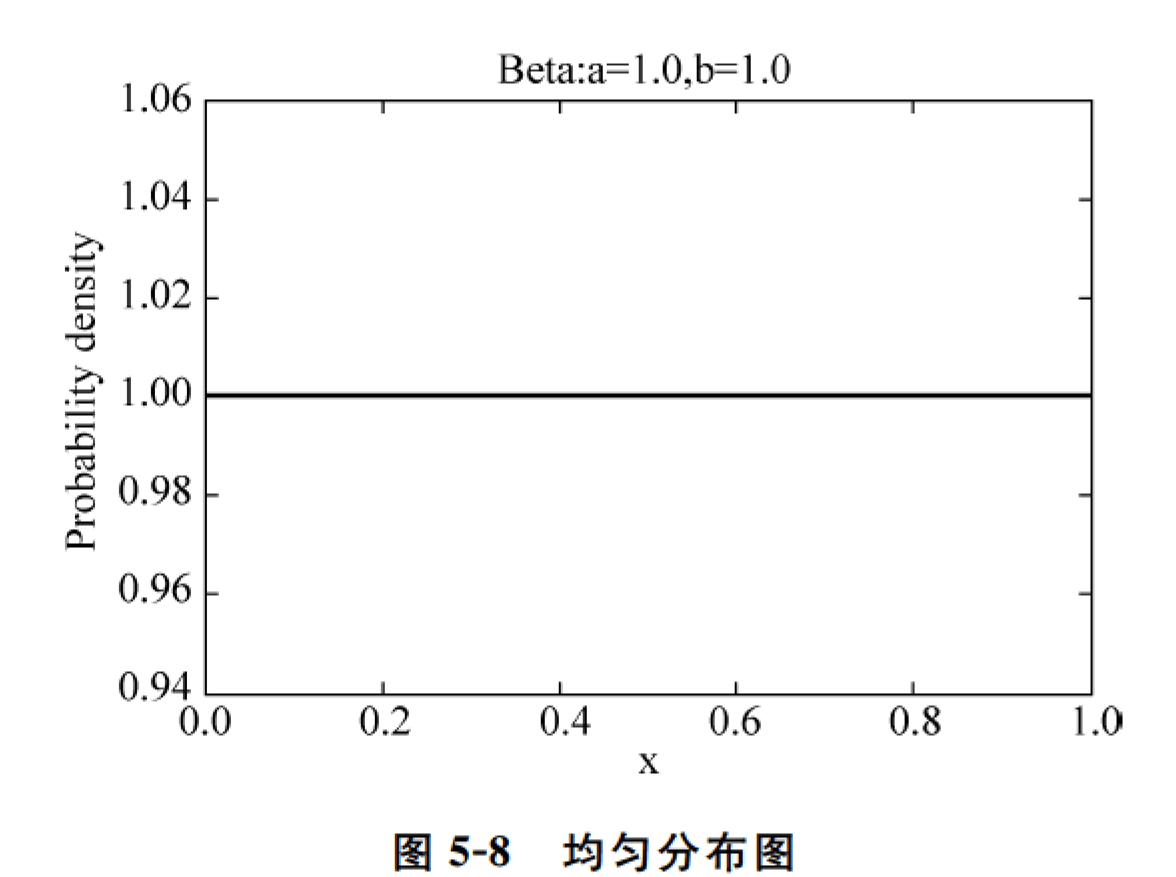


图 5-8　均匀分布图

```
data = randint(0, 10, size = 10)
#data = stats.norm.rvs(loc = 0.0, scale = 1.0, size = 1000)
print "Mean: %g" % np.mean(data)
print "SD: %g" % np.std(data,ddof = 1)
```

```
plt.hist(data,bins = 20,normed = True)
plt.xlabel("numbers of accidents")
plt.ylabel("Norm distribution")
plt.show()
```

5.6 指数分布

指数分布(exponential distribution)是一种连续概率分布,用于表示独立随机事件发生的时间间隔。比如旅客进入机场的时间间隔、打进客服中心电话的时间间隔、中文维基百科新条目出现的时间间隔等。

$$P(x;\lambda)=\lambda e^{-\lambda x},\quad E(X)=1/\lambda,\quad \mathrm{var}(X)=1/\lambda^2$$

将参数 λ 设置为 0.5,并将 x 的取值范围设置为[0,15]。Python 代码如下:

```
lambd = 0.5
x = np.arange(0,15,0.1)
y = lambd * np.exp( - lambd * x)
plt.plot(x,y)
plt.title('Exponential:: $\lambda $ = %.2f' % lambd)
plt.xlabel("x")
plt.ylabel("Probability density")
plt.show()
```

执行上述代码,得到如图 5-9 所示的结果。

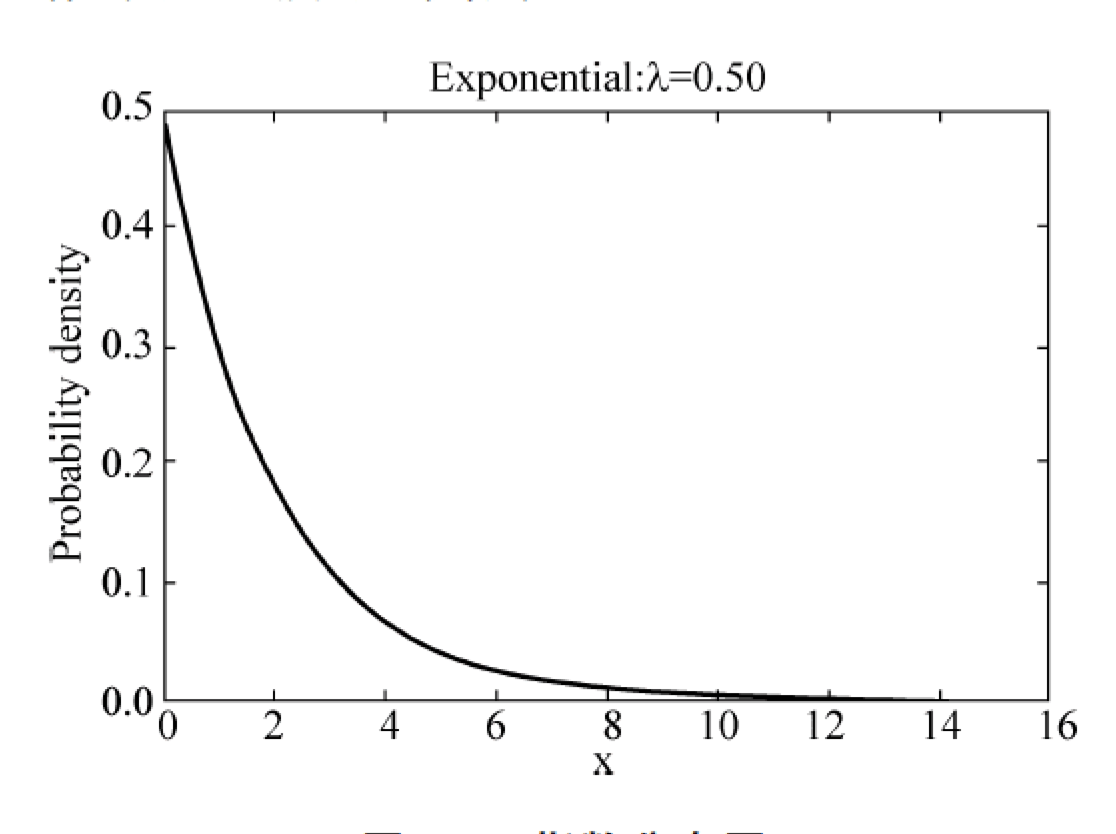


图 5-9 指数分布图

接着,在指数分布下模拟 1000 个随机变量。scale 参数表示 λ 的倒数。函数 np.std 中,参数 ddof 等于标准偏差除以 $n-1$ 的值。

Python 代码如下:

```
data =  stats.norm.rvs(loc = 0.0, scale = 1.0, size = 1000)
print "Mean: %g" % np.mean(data)
print "SD: %g" % np.std(data,ddof = 1)
plt.hist(data,bins = 20,normed = True)
plt.xlabel("numbers of accidents")
plt.ylabel("Norm distribution")
plt.show()
```

执行上述代码后，将输出这些随机变量的平均值和标准差，然后画出所有的随机变量的直方图，如图 5-10 所示。

```
Mean:1.92958
SD:1.85039
```

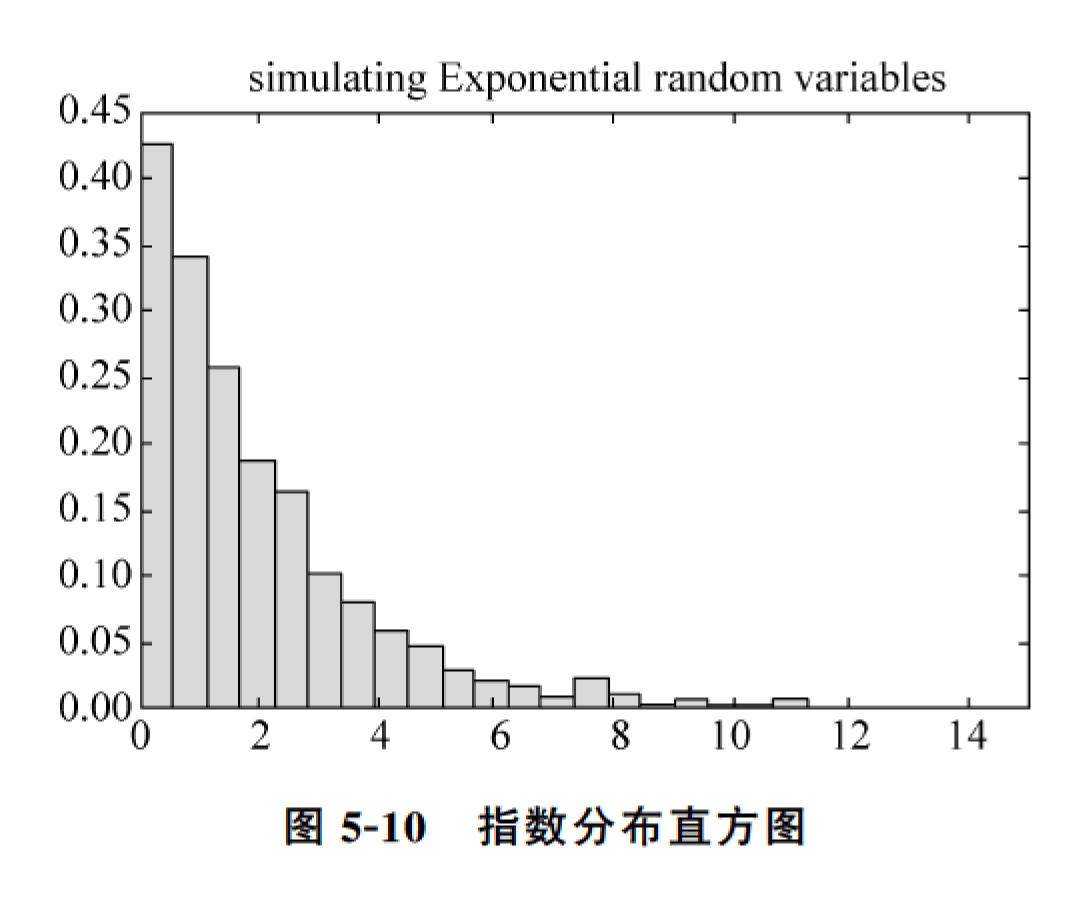


图 5-10　指数分布直方图

练　习　题

对本章的各种分布尝试不同的参数绘制图形。



描述性统计的 Python 应用

统计就是搜集数据，让我们知道总体状况怎么样。它更重要的意义在于数据分析，即作出判断和预测。

描述性统计是对数据的性质进行描述，例如：均值描述了数据的中心趋势，方差描述了数据的离散程度。

推断统计是用来作判断和预测的。例如，假设检验就是用来作判断的，回归分析和时间序列分析是用来作预测的。

6.1 描述性统计量

6.1.1 总体和样本

总体是我们所要研究的所有个体的集合。如中国人的身高集合就是一个总体，从中抽取 100 人的身高就是一个样本。

我们研究一个总体，通常不是要了解每一个个体的情况，而是想要知道某些总体参数。例如想中国人的平均身高是多少，这样就可以与 10 年前的平均身高作比较。

但由于种种原因，我们通常不能得到总体中所有个体的数值，而只能抽取一个样本，来计算样本统计量。样本统计量是样本中个体数值的函数，例如样本均值、样本方差等。例如我们抽取 100 个中国人，分别量了他们的身高，计算了他们的平均身高，用来估计中国人总体的平均身高。统计过程用图形表示如图 6-1 所示。

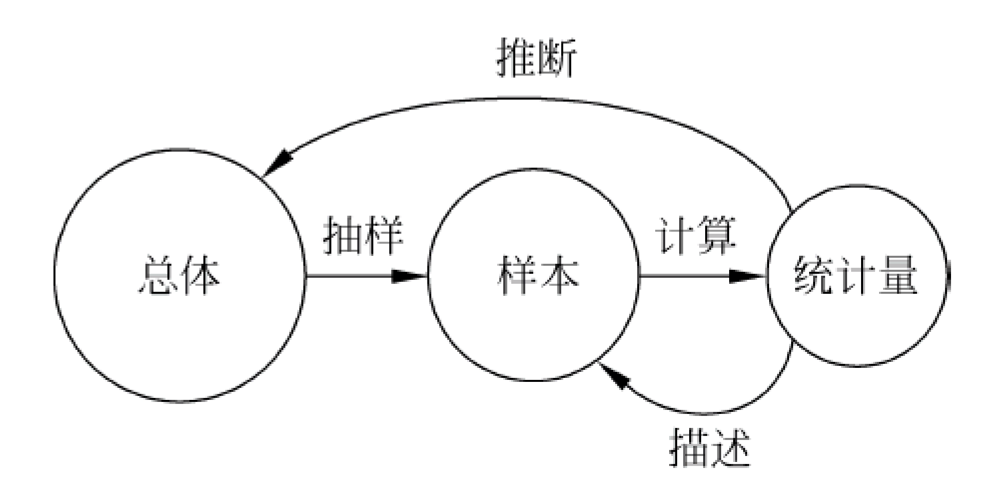


图 6-1 统计过程

6.1.2 度量尺度

为了选择一个恰当的统计方法来描述和分析数据，我们需要区分不同的度量尺度(或测量标准)。数据尺度有强有弱，但不外乎 4 种：名义、顺序、区间和比率。

名义尺度：代表最简单的度量标准，它对数据进行分类但不进行排序。如用 1 表示男，0 表示女。

顺序尺度：代表稍微强一点的度量标准，它根据某种特征排序，将数据分成不同类别。

间隔尺度：它比排序尺度更进一步，使得数据之间间隔相等。不仅能比较大小，还能做加减运算，但不能做乘除运算。例如，上海温度是 20℃，北京是 10℃，可以说上海温度比北京温度高 10℃，但不能说上海的温度是北京的两倍。

比例尺度。它比间隔尺度更进一步，它增加了一个绝对零点，不仅能比较大小，能做加减运算，还能做乘除运算。例如人的身高、债券的价格等。

以上 4 种度量尺度是按照由弱到强的顺序排列的。

6.1.3 频数分布

频数分布是指一种用表格列示数据的方法，它用较少的区间对数据总体进行概括。实际落入一个给定区间的观测值数量称为绝对频数，或简称频数。每个区间的绝对频数除以整个样本观测值的数量。

建立一个频数分布的基本步骤如下：

(1) 将数据以升序排列。

(2) 计算数据的极差，定义极差＝最大－最小。

(3) 确定频数分布包含的区间数 k。

(4) 确定区间的宽度(极差/k)。

(5) 不断地在数据最小值上加上区间宽度来确定各个区间的端点，此过程在到达包含最大值的区间时停止。

(6) 计算落入每个区间中观测值的个数。

(7) 建立一个列示落入从小到大排列的每个区间中观测值数量的表格。

例如，某股票过去 25 年的年收益率如下(通过排序)：

－28％，－22％，－19％，－18％，－12％，－9％，－8％，－6％，－1％，1％，2％，3％，4％，5％，6％，7％，11％，15％，16％，17％，18％，20％，23％，26％，38％。

现在我们看看这个股票的收益率分布情况。

我们发现，收益率位于－30％～40％之间。将－30％～40％区间分段，每 10％为一段，共分 7 段。

最后得到的结果如表 6-1 所示。

表 6-1 频 数 表

区 间 段	绝对频数	相对频数	累积绝对频数	累积相对频数
[－30％，－20％)	2	0.08	2	0.08
[－20％，－10％)	3	0.12	5	0.2
[－10％，－0％)	4	0.16	9	0.36
[0％，10％)	7	0.28	16	0.64
[10％，20％)	5	0.2	21	0.84
[20％，30％)	3	0.12	24	0.96
[30％，40％]	1	0.04	25	1
总计	25	1		

频数数据的柱状图如图 6-2 所示。

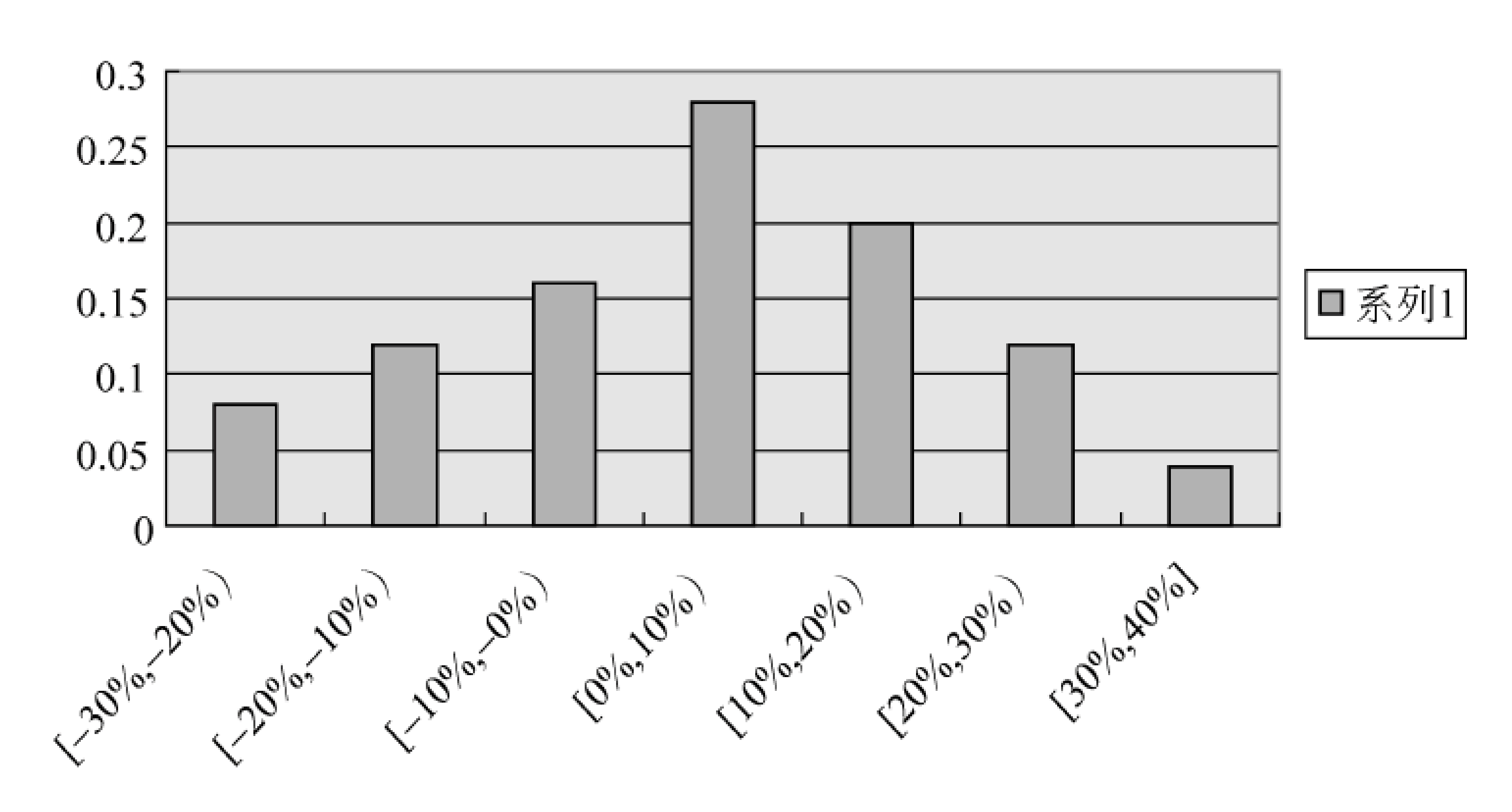


图 6-2　频数数据的柱状图

相对频数的折线图如图 6-3 所示。

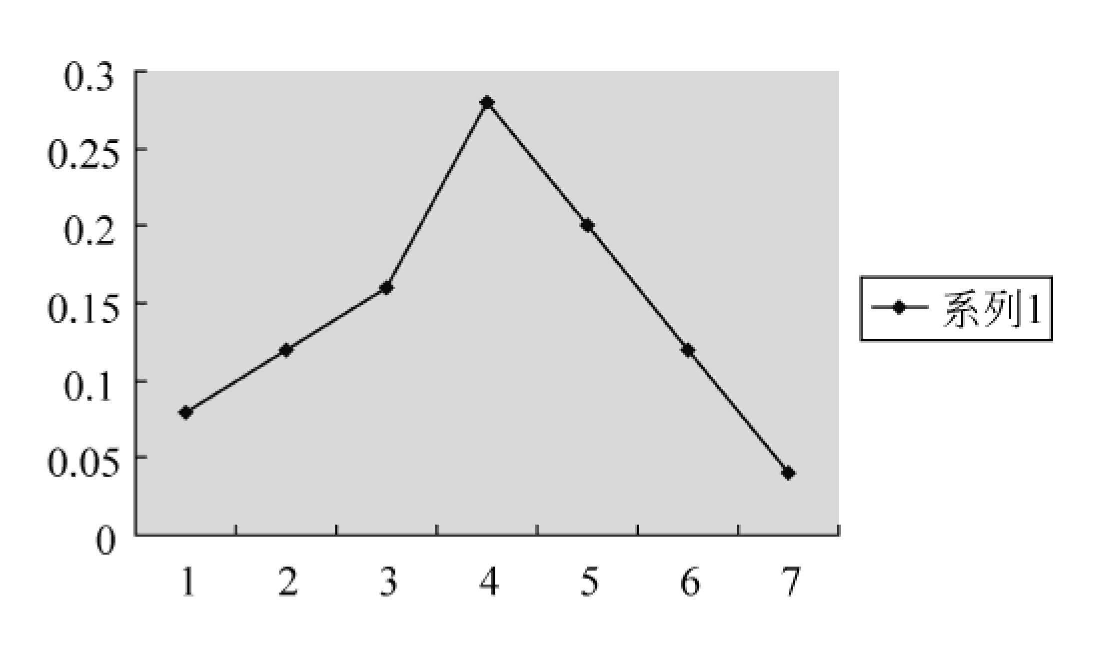


图 6-3　相对频数的折线图

6.1.4　集中趋势的度量

拿到一组数据，我们首先想知道这组数据的中心位置在哪里，即数据围绕什么中心数值波动，这称为集中趋势的度量。通常用均值来度量，均值有如下 4 种。

1. 算术平均

总体均值
$$\mu = \frac{1}{N}\sum_{i=1}^{N} X_i$$

样本均值
$$\bar{x} = \frac{1}{n}\sum_{i=1}^{N} x_i$$

2. 几何平均

$$\bar{x}_g = \sqrt[n]{x_1 x_2 \cdots x_n}$$

在金融学中的绩效平均时，历年收益率的平均收益率应该用几何平均率，即时间加权收益率，它不受投资项目资金流入和流出的影响。几何平均收益率为 t 年收益率分别加 1 之后相乘，再开 t 次方，然后减去 1。公式为：

$$\bar{R}_g = \sqrt[t]{(1+R_1)(1+R_2)\cdots(1+R_n)} - 1$$

3. 加权平均

$$\bar{x}_w = \sum_{i=1}^{n} w_i x_i$$

其中 w_i 为 x_i 的权重，且权重之和为 1。当所有权重相等时，加权平均即为算术平均。加权平均在金融学中的应用：一个资产组合的收益率，等于其中各个资产收益率的加权平均，权重为各个资产市值占总资产组合市值的百分比。

4. 调和平均

$$\bar{x}_h = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

当观测值不全相等时，有：调和平均<几何平均<算术平均。

6.1.5 中位数

如果有一组数据，把它按从小到大的顺序排列，将这一数列等分成两份，这个分位数称为中位数，对于奇数个数组成的数列，中位数就是中间的那个数，对于偶数个数组成的数列，中位数就是中间的那个两个数相加除以 2。

由于均值受异常值的影响较大，因此用均值来估计中心趋势显得很不稳定，而中位数的优点是受异常值影响较小，估计量稳定。

6.1.6 众数

众数就是一组数据中出现次数最多的数。

如数列：1，1，2，2，3，3，3，4，5，其众数为 3。

如数列：1，1，1，2，2，3，3，3，4，5，其众数为 1 和 3。

如数列：1，2，3，4，5，没有众数。

一组数据可能有一个众数，可能有多个众数，也可能没有。众数的这一性质使得其使用范围受到限制。

6.1.7 分位数

如果我们有一组数据，把它们按从小到大的顺序排列，分位数就是正好能将这一数列等分的数。

将这一数列等分成两份，这个分位数称为中位数。将这一数列等分为 4 份，这 3 个分位数都称为四分位数，它从小到大依次称作：第 1 个四分位数、第 2 个四分位数、第 3 个四分位数。第 2 个四分位数就是中位数。

也可以将这一数列等分成 5 份，得到 4 个五分位数。也可以将这一数列等分成 10 份，得到 9 个十分位数。也可以将这一数列等分成 100 份，得到 99 个百分位数。

我们可以把所有的分位数都转换成百分数。例如，第 2 个五分位数就是第 40 个百分位数，第 3 个四分位数就是第 75 个百分位数。这样，我们就可以用以下公式来计算分位数：

$$L_y = (n+1)y/100$$

其中，

n：数列中一共有多少个数；

y：第几个百分数；

L_y：结果是数列的第几个数。

例如：

有这样一组数：2,5,7,9,12,16,21,34,39,计算第 4 个五分位数。

第 4 个五分位数就是第 80 个百分位数,数列共有 9 个数,套用公式

$$L_y = (n+1)y/100 = (9+1) \times 80/100 = 8$$

数列的第 8 个数即为 34。

有这样一组数列：2,5,7,9,12,16,21,34,39,40,计算第 4 个五分位数。

第 4 个五分位数就是第 80 个百分位数,数列共有 10 个数,套用公式

$$L_y = (n+1)y/100 = (10+1) \times 80/100 = 8.8$$

数列的第 8.8 个数是什么意思,就是第 8 个数再往右的 0.8 个数,第 8 个数是 34,第 9 个数是 39,相差 5,那么 0.8 个数就是 5×0.8=4,所以 34+4=38,即第 4 个五分位数是 38。

6.1.8 离散程度的度量

知道了一组数据的中心位置之后,就想知道数据距离中心位置是远还是近,这称为离散程度的度量。在金融分析中,常用离散程度来衡量风险。

1. 极差

极差定义为

$$\text{极差} = \text{最大值} - \text{最小值}$$

极差越小,离散程度越小。由定义可知极差只用到了一组数据中的两个数据,而忽略了数据的分布状况等许多有用的信息,因此仅仅用极差来度量离散程度显得很不够。

2. 平均绝对差

平均绝对差定义为

$$\text{MAD} = \frac{\sum_{i=1}^{n} |x_i - \bar{x}|}{n}$$

式中$\bar{x}$表示样本的均值,n 表示样本中观测值的数目。

3. 总体方差和总体标准差

总体方差定义为

$$\sigma^2 = \frac{\sum_{i=1}^{N} (X_i - \mu)^2}{N}$$

式中,μ 表示总体均值,N 表示总体的规模。

总体标准差定义为

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(X_i - \mu)^2}{N}}$$

式中，μ 表示总体均值，N 表示总体的规模。

4. 样本方差和样本标准差

样本方差定义为

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

式中，$\bar{x}$ 表示总体均值，n 表示总体的规模。

样本标准差定义为

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

式中，$\bar{x}$ 表示总体均值，n 表示总体的规模。

5. 变异系数

变异系数 CV 定义为标准差除以均值。用公式表示为

$$\mathrm{CV} = \frac{s}{\bar{x}}$$

s 与 $\bar{x}$ 的含义如上所示。

6. 偏度

偏度是衡量一组数据左右偏离的程度。

左右对称的分布偏度为 0。左右对称的分布，其均值、中位数和众数相等。如图 6-4 所示是一个对称的分布。

如图 6-5 所示是一个非对称的右偏分布。在右偏分布中，均值大于中位数大于众数。

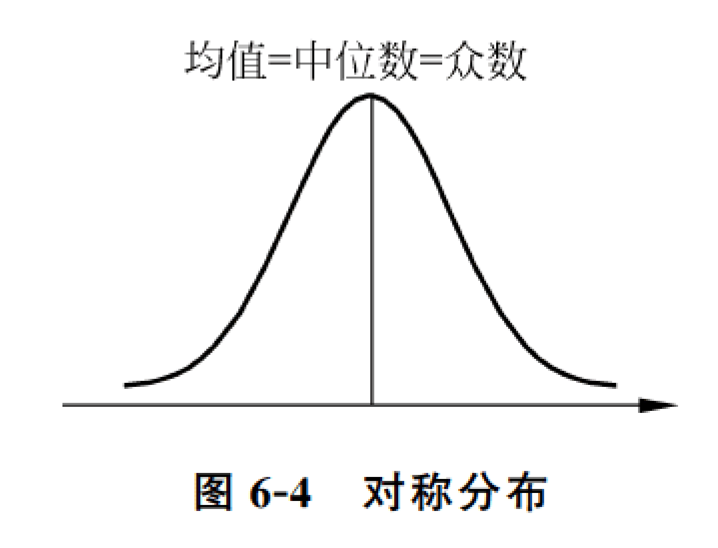


图 6-4　对称分布

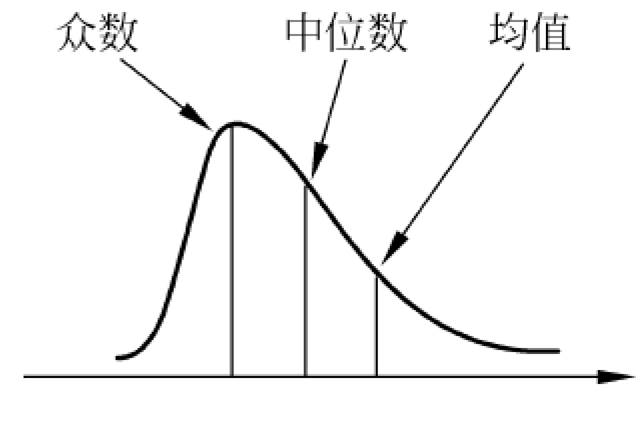


图 6-5　右偏(正偏)分布图

如图 6-6 所示是一个非对称的左偏分布。在左偏分布中，均值小于中位数小于众数。

7. 峰度

峰度是衡量一组数据峰值高于或低于正态分布的程度。任何一个正态分布的峰度为

3。如果一个分布的峰度大于 3 称为高峰态,小于 3 称为低峰态。

常把峰度的数值减去 3,称为超额峰度。同样,任何一个正态分布的超额峰度为 0。如果一个分布的超额峰度大于 0 称为高峰态,小于 0 称为低峰态。

低峰态、高峰态与正态分布的对比如图 6-7 所示(虚线为正态分布)。

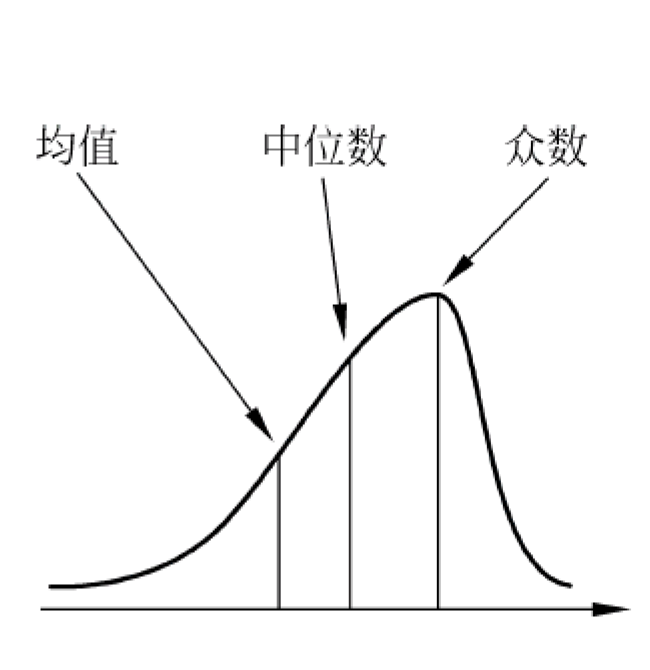


图 6-6 左偏(负偏)分布图

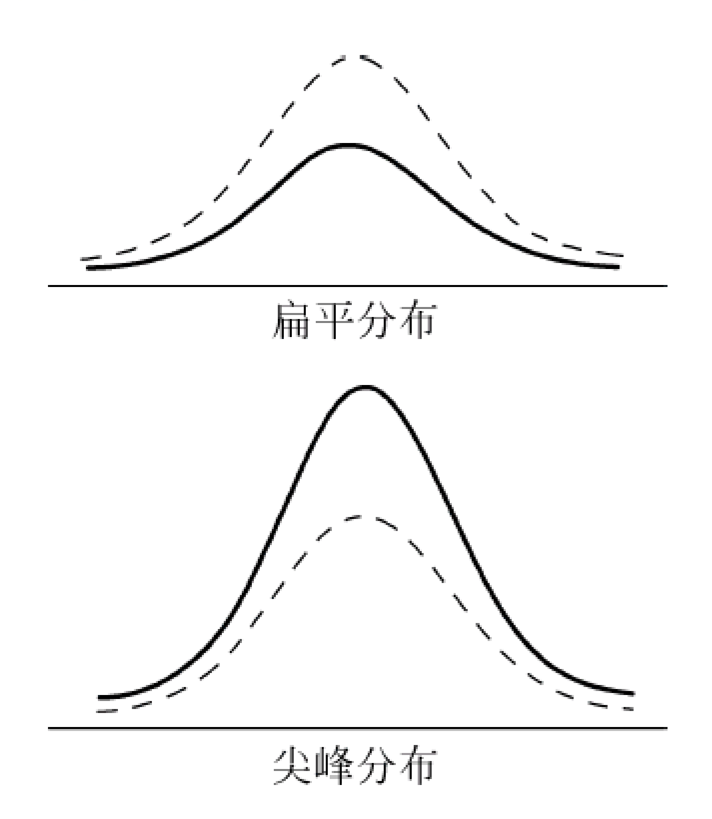


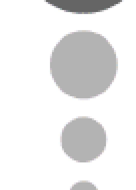
图 6-7 低峰态、高峰态与正态分布的对比图

6.2 描述性统计的 Python 工具

Python 中的 Pandas 常用的统计方法如表 6-2 所示。

表 6-2 Pandas 常用的统计方法

函数名称	作用
count	非 NA 值的数量
describe	针对 Series 或 DF 的列计算汇总统计
min,max	最小值和最大值
argmin,argmax	最小值和最大值的索引位置(整数)
idxmin,idxmax	最小值和最大值的索引值
quantile	样本分位数(0 到 1)
sum	求和
mean	均值
median	中位数
mad	根据均值计算平均绝对离差
var	方差
std	标准差
skew	样本值的偏度(三阶矩)
kurt	样本值的峰度(四阶矩)
cumsum	样本值的累计和
cummin,cummax	样本值的累计最大值和累计最小值
cumprod	样本值的累计积
diff	计算一阶差分(对时间序列很有用)
pct_change	计算百分数变化



Python 中 NumPy 和 SciPy 常用的统计方法如表 6-3 所示。

表 6-3　NumPy 和 SciPy 常用的统计方法

程　序　包	方　　法	说　　明
NumPy	array	创造一组数
NumPy. random	normal	创造一组服从正态分布的定量数
NumPy. random	randint	创造一组服从均匀分布的定性数
NumPy	mean	计算均值
NumPy	median	计算中位数
SciPy. stats	mode	计算众数
NumPy	ptp	计算极差
NumPy	var	计算方差
NumPy	std	计算标准差
NumPy	cov	计算协方差
NumPy	corrcoef	计算相关系数

6.3　单组数据描述性统计的 Python 应用

我们知道，样本来自总体，样本的观测值中含有总体各方面的信息，但这些信息较为分散，有时显得杂乱无章。为将这些分散在样本中的有关总体的信息集中起来以反映总体的各种特征，需要对样本进行加工得到统计量。均值、标准差、五数(最小值、第三四分位数、中位数，第一四分位数、最大值)是数据分析的主要的统计量，它们对数据的进一步分析很有帮助。

6.3.1　总体描述

例 6-1　为了解我国各地区的电力消费情况，某课题组搜集整理了 2009 年我国各省市的电力消费的数据，如表 6-4 所示。试通过对数据进行基本描述性分析来了解我国各省市的电力消费情况。

表 6-4　2009 年我国各个省市的电力消费情况

地　　区	电 力 消 费
北京	739.146
天津	550.156
河北	2343.85
山西	1267.54
内蒙古	1287.93
……	…
青海	337.24
宁夏	462.96
新疆	547.88

在目录 G:\2glkx\data 下建立 al6-1. xls 数据文件后，从 Excel 取数的 Python 代码如下：

```
import Pandas as pd
df = pd.read_excel('G:\\2glkx\\data\\al6 - 1.xls')
```

```
df.head()
```

可得到前 5 条记录的数据如下：

```
            region  consumption
0          Beijing   739.146484
1          Tianjin   550.155579
2            Hebei  2343.846680
3           Shanxi  1267.537598
4   Inner Mongolia  1287.925659
```

输入如下命令：

```
df.describe()
```

得到如下描述性统计结果：

```
     consumption
count    31.000000
mean   1180.488799
std     903.556130
min      17.698700
25%     579.689575
50%     891.190186
75%    1306.267822
max    3609.642334
```

如果只需要 consumption 的均值，可以利用函数 df.mean()实现。

```
df.mean()
consumption    1180.488799
```

6.3.2 样本分位数与众数、最大、最小值描述

若要得到 consumption 分位数用 df.quantile()，例如要得到 0%，25%，50%，75%，100%的分位数，可用：

```
df.quantile(0),df.quantile(0.25),df.quantile(0.50),df.quantile(0.75),df.quantile(1.0)
```

结果如下：

```
(consumption    17.6987
 dtype: float64, consumption    579.689575
 dtype: float64, consumption    891.190186
 dtype: float64, consumption    1306.267822
 dtype: float64, consumption    3609.642334
 dtype: float64)
df.median()           #计算中位数
consumption    891.190186
df.max()              #求最大值
    consumption       3609.64
    最小值使用 df.min()。
df.min()              #求最小值
consumption    17.6987
```

6.3.3　离差描述

样本的平均水平可以用上面介绍的平均值函数 mean()和中位数函数 median()来计算。样本的变异程度可以用标准差函数(sd())、方差函数(var())和绝对离差函数(mad())来表示。方差函数 var()也可用于计算两个向量协方差或一个矩阵的协方差阵。对于 $x=(x_1,\cdots,x_n)$,sd()的定义为

$$\mathrm{sd}(x)=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$$

```
df.std()              #求 consumption 标准差
consumption     903.55613
df.var()              #求 consumption 方差
consumption     816413.679584
df.mad()              #根据均值计算平均绝对离差
consumption     666.110451
```

6.3.4　偏度与峰度描述

偏度：
$$\beta_1=\frac{E(X-E(X))^3}{(E[X-E(X)]^2)^{3/2}}$$

峰度：
$$\beta_1=\frac{E(X-E(X))^4}{(E[X-E(X)]^2)^{4/2}}-3$$

Python 的 Pandas 程序包提供了求偏度 skew()、峰度 kurt()的函数。

```
df.skew()             #求偏度
consumption     1.376556
df.kurt() - 3         #求峰度
consumption     1.272889
```

6.3.5　使用 NumPy 和 SciPy 进行描述性统计

```
import NumPy as np
import Pandas as pd
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al6 - 1.xls'))
data.head()
```

得到前 5 条记录如下：

```
              region  consumption
0            Beijing   739.146484
1            Tianjin   550.155579
2              Hebei  2343.846680
3             Shanxi  1267.537598
4     Inner Mongolia  1287.925659
x = np.array(data[['consumption']])
sum(x)
Out[10]: array([ 36595.15275574])
from SciPy import mean
```

```
mean(x)
Out[14]: 1180.4887985721712
from SciPy import median
median(x)
Out[18]: 891.190185546875
from SciPy import var
var(x)
Out[21]: 790077.75443609583
from SciPy import std
std(x)
Out[23]: 888.86318094299293
from SciPy.stats import skew
skew(x)
Out[28]: array([ 1.30903196])
```

6.4 多组数据描述性统计的 Python 应用

6.4.1 多组数据描述性统计的 Python-Pandas 应用

对多组数据进行描述性统计与单组数据类似，直接使用 Pandas 的 describe()可以得到各组数据的描述性统计数据。先看一个例子。

例 6-2 数据见表 6-5。

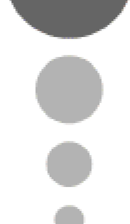
表 6-5 多组数据

BH	Z1	Z2	Z3	Z4	K
1	7	26	6	60	78.5
2	1	29	15	52	74.3
3	11	56	8	20	104.3
4	11	31	8	47	87.6
5	7	52	6	33	95.9
6	11	55	9	22	109.2
7	3	71	17	6	102.7
8	1	31	22	44	72.5
9	2	54	18	22	93.1
10	21	47	4	18	115.9
11	1	40	23	34	83.8
12	11	66	9	12	113.3
13	10	68	8	12	109.4

```
import Pandas as pd
df = pd.read_excel('G:\\2glkx\\data\\al3 - 1.xls')
df.head()
```

数据 df 描述了 BH，Z1，Z2，Z3，Z4，K 数据，前 5 条记录数据如下：

```
   BH  Z1  Z2  Z3  Z4     K
0   1   7  26   6  60  78.5
1   2   1  29  15  52  74.3
```




```
2    3   11   56    8   20   104.3
3    4   11   31    8   47    87.6
4    5    7   52    6   33    95.9
```

使用函数 describe()描述 al3-1.xls 的结果如下：

```
df.describe()
              BH          Z1          Z2          Z3          Z4           K
count   13.00000   13.000000   13.000000   13.000000   13.000000   13.000000
mean     7.00000    7.461538   48.153846   11.769231   29.384615   95.423077
std      3.89444    5.882394   15.560881    6.405126   17.041804   15.043723
min      1.00000    1.000000   26.000000    4.000000    6.000000   72.500000
25%      4.00000    2.000000   31.000000    8.000000   18.000000   83.800000
50%      7.00000    7.000000   52.000000    9.000000   22.000000   95.900000
75%     10.00000   11.000000   56.000000   17.000000   44.000000  109.200000
max     13.00000   21.000000   71.000000   23.000000   60.000000  115.900000
```

从上面数据可以看出，df.describe()描述统计了 BH，Z1，Z2，Z3，Z4，K 等变量的计数、均值、标准差、最小值、25%的分位数、50%的分位数、75%的分位数、最大值等。

6.4.2　方差、协方差计算的 Python-NumPy 应用

Python 中 NumPy 的 var、cov 用来计算多个变量的方差、协方差。

```
import NumPy as np
df.var()
```

得到如下结果：

```
BH     15.166667
Z1     34.602564
Z2    242.141026
Z3     41.025641
Z4    290.423077
K     226.313590
dtype: float64
df.cov()
```

得到协方差矩阵如下：

```
            BH          Z1           Z2          Z3           Z4           K
BH   15.166667    3.166667    33.416667    5.583333   -43.250000   28.416667
Z1    3.166667   34.602564    20.923077  -31.051282   -33.192308   64.663462
Z2   33.416667   20.923077   242.141026  -13.878205  -252.647436  191.079487
Z3    5.583333  -31.051282   -13.878205   41.025641     8.346154  -51.519231
Z4  -43.250000  -33.192308  -252.647436    8.346154   290.423077 -220.459615
K    28.416667   64.663462   191.079487  -51.519231  -220.459615  226.313590
```

练　习　题

对本章例题，使用 Python 重新操作一遍。

第7章 参数估计的 Python 应用

7.1　参数估计与置信区间的含义

根据样本推断总体的分布和分布的数字特征称为统计推断。本章我们来讨论统计推断的一个基本问题——参数估计。参数估计有两类，一类是点估计，就是以某个统计量的样本观察值作为未知参数的估计值；另一类是区间估计，就是用两个统计量所构成的区间来估计未知参数。我们在估计总体均值的时候，用样本均值作为总体均值的估计，就是点估计。在做置信区间估计之前，必须先规定一个置信度，例如 95%。置信度以 1-alpha 表示，这里的 alpha 就是假设检验里的显著性水平。因此 95%的置信度就相对于 5%的显著性水平。

置信区间估计的一般公式为：点估计±关键值×样本均值的标准误

$$\bar{x} \pm z_{\alpha/2} \cdot \frac{s}{\sqrt{n}}$$

这里的关键值就是以显著性水平 α 做双尾检验的关键值。关键值是 z 关键值或 t 关键值。究竟是 z 关键值还是 t 关键值，如表 7-1 所示。

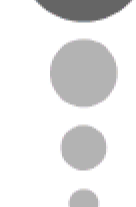
表 7-1　z 关键值与 t 关键值选择

项　　目	正态总体 $n<30$	$n>=30$
已知总体方差	z	z
未知总体方差	t	t 或 z

假设一位投资分析师从股权基金中选取了一个随机样本，并计算出了平均的夏普比率。样本的容量为 100，并且平均的夏普比率为 0.45。该样本具有的标准差为 0.30。利用一个基于标准正态分布的临界值，计算并解释所有股权基金总体均值的 90%置信区间。这个 90%的置信区间的临界值为 $z_{0.05}=1.65$，故置信区间为 $\bar{x} \pm z_{0.05}\frac{s}{\sqrt{n}}=0.45 \pm 1.65 \times \frac{0.30}{\sqrt{100}}$，即 0.4005～0.4495，分析师可以说有 90%的信心认为这个区间包含了总体均值。

7.2　点估计的 Python 应用

由大数定律可知，如果总体 X 的 k 阶矩存在，则样本的 k 阶矩以概率收敛到总体的 k 阶矩，样本矩的连续函数收敛到总体矩的连续函数。这就启发我们可以用样本矩作为总体



矩的估计量，这种用相应的样本矩去估计总体矩的估计方法称为矩估计法。

设 $X_1,\cdots,X_n$ 为来自某总体的一个样本，样本的 k 阶原点矩为

$$A_k = \frac{1}{n}\sum_{i=1}^{n} X_i^k,\quad k = 1,2,\cdots$$

如果总体 X 的 k 阶原点矩 $\mu_k=E(X^k)$存在，则按矩法估计的思想，用 A_k 去估计 μ_k。$\hat{\mu}_k=A_k$。

设总体 X 的分布函数含有 k 个未知参数 $\theta=(\theta_1,\cdots,\theta_k)$，$j=1,2,\cdots,k$，且分布的前 k 阶矩存在，它们都是 $\theta_1,\cdots,\theta_k$ 的函数，此时求 $\theta_j(j=1,2,\cdots,k)$的矩估计的步骤如下。

（1）求出 $E(X^j)=\mu_j$，$j=1,2,\cdots,k$，并假定

$$\mu_j = g_j(\theta_1,\cdots,\theta_k),\quad j = 1,2,\cdots,k \tag{1}$$

（2）解方程组(1)得到

$$\theta_i = h_i(\mu_1,\cdots,\mu_k),\quad i = 1,2,\cdots,k \tag{2}$$

（3）在上式中用 A_j 代替 μ_j，$j=1,2,\cdots,k$ 即得 $\theta=(\theta_1,\cdots,\theta_k)$的矩估计

$$\hat{\theta}_i = h_i(A_1,\cdots,A_k),\quad i = 1,2,\cdots,k \tag{3}$$

若有样本观察值 $x_1,\cdots,x_n$，代入上式即可得到 $\theta=(\theta_1,\cdots,\theta_k)$的估计值。

由于函数 g_j 的表达式不同，求解上述方程或方程组会相当困难，这时需要应用迭代算法进行数值求解。这需要具体问题具体分析。我们不可能有固定的 R 语言程序来直接估计 θ，只能利用 R 的计算功能根据具体问题编写相应的 R 程序，下面看一个例子。

例 7-1　设 $X_1,\cdots,X_n$ 为来自 $b(1,\theta)$的一个样本，θ 表示某事件的成功的概率，通常事件的成败机会比 $g(\theta)=\theta/(1-\theta)$是人们感兴趣的参数，可以利用矩估计轻松给出 $g(\theta)$一个很不错的估计。因为 θ 是总体均值，由矩法，记 $\bar{X}=\frac{1}{n}\sum_{i=1}^{n}X_i$，则 $h(\bar{X})=\frac{\bar{X}}{1-\bar{X}}$ 是 $g(\theta)$ 的一个矩估计。

例 7-2　对某个篮球运动员记录其在某一次比赛中投篮命中与否，观测数据如下：

1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 1

0 0 1 0 1 0 1 0 0 1 1 0 1 1 0 1

编写 Python 程序估计这个篮球运动员投篮的成败比。

```
import numpy as np
x = [1,1,0,1,0,0,1,0,1,1,1,0,1,1,0,1,0,0,1,0,1,0,1,0,0,1,1,0,1,1,0,1]
theta = np.mean(x)
h = theta/(1 - theta)
print 'h = ',h
h =  1.28571428571
```

我们得到 $g(\theta)$的矩估计为 1.28571428571。

7.3　单正态总体均值区间估计的 Python 应用

上一节讨论了点估计，由于点估计值只是估计量的一个近似值，因而点估计本身既没有反映出这种近似值的精度，即指出用估计值去估计的误差范围有多大，也没有指出这个误差范围以多大的概率包括未知参数，这正是区间估计要解决的问题。本节讨论单正态总体均值的区间估计问题。

7.3.1 方差 $\sigma_0=\sigma$ 已知时 μ 的置信区间

设来自正态总体 $N(\mu,\sigma^2)$ 的随机样本和样本值记为 $X_1,X_2,\cdots,X_n$，样本均值 $\overline{X}$ 是总体均值 μ 的一个很好的估计量，利用 $\overline{X}$ 的分布，可以得出总体均值 μ 的置信度为 $1-\alpha$ 的置信区间(通常取 $\alpha=0.05$)。

由于 $\overline{X}\sim N(\mu,\sigma^2)$，因此有 $Z=\dfrac{\overline{X}-\mu}{\sigma/\sqrt{n}}\sim N(0,1)$。

由 $P(-z_{1-\alpha/2}<Z<z_{1-\alpha/2})=1-\alpha$ 即得

$$P\left(\overline{X}-\frac{\sigma}{\sqrt{n}}z_{1-\alpha/2}<\mu<\overline{X}+\frac{\sigma}{\sqrt{n}}z_{1-\alpha/2}\right)=1-\alpha$$

所以对于单个正态总体 $N(\mu,\sigma^2)$，当 $\sigma_0=\sigma$ 已知时，μ 的置信区间为 $1-\alpha$ 的置信区间为 $\left(\overline{X}-\dfrac{\sigma}{\sqrt{n}}z_{1-\alpha/2},\overline{X}+\dfrac{\sigma}{\sqrt{n}}z_{1-\alpha/2}\right)$。

同理可求得 μ 的置信度为 $1-\alpha$ 的置信上限为 $\overline{X}+\dfrac{\sigma}{\sqrt{n}}z_{1-\alpha}$。

μ 的置信度为 $1-\alpha$ 的置信下限为 $\overline{X}-\dfrac{\sigma}{\sqrt{n}}z_{1-\alpha}$。

例 7-3 某车间生产的滚珠直径 X 服从正态分布 $N(\mu,0.6)$。现从某天的产品中抽取 6 个，测得直径如下(单位：mm)。

14.6，15.1，14.9，14.8，15.2，15.1

试求平均直径置信度为 95%的置信区间。

解：置信度 $1-\alpha=0.95$，$\alpha=0.05$。$\alpha/2=0.025$，查表可得 $Z_{0.025}=1.96$，又由样本值得 $\bar{x}=14.95$，$n=6$，$\sigma=\sqrt{0.6}$。由上式有

置信下限 $\bar{x}-Z_{1-\alpha/2}\dfrac{\sigma_0}{\sqrt{n}}=14.95-1.96\times\sqrt{\dfrac{0.6}{6}}=14.3302$

置信上限 $\bar{x}+Z_{1-\alpha/2}\dfrac{\sigma_0}{\sqrt{n}}=14.95+1.96\times\sqrt{\dfrac{0.6}{6}}=15.5698$

所以均值的置信区间为(14.3302，15.5698)。

为此，我们编写的 Pyhton 程序如下：

```
import numpy as np
import scipy.stats as ss
n = 6; p = 0.025; sigma = np.sqrt(0.6)
x = [14.6,15.1,14.9,14.8,15.2,15.1]
xbar = np.mean(x)
low = xbar - ss.norm.ppf(q = 1 - p) * (sigma / np.sqrt(n))
up = xbar + ss.norm.ppf(q = 1 - p) * (sigma / np.sqrt(n))
print 'low = ',low
print 'up = ',up
low = 14.3302049677
up = 15.5697950323
```

7.3.2 方差 σ^2 未知时 μ 的置信区间

由于
$$Z=\frac{\overline{X}-\mu}{\sigma/\sqrt{n}}\sim N(0,1),\frac{(n-1)S^2}{\sigma^2}\sim\chi^2(n-1)$$

且二者独立，所以有

$$T=\frac{\overline{X}-\mu}{S/\sqrt{n}}\sim t(n-1)$$

同样由 $P(-t_{1-\alpha/2}(n-1)<T<t_{1-\alpha/2}(n-1))=1-\alpha$ 得到

$$P\left(\overline{X}-\frac{S}{\sqrt{n}}t_{1-\alpha/2}(n-1)<\mu<\overline{X}+\frac{S}{\sqrt{n}}t_{1-\alpha/2}(n-1)\right)=1-\alpha$$

所以方差 σ^2 未知时 μ 的置信度为 $1-\alpha$ 的置信区间为

$$\left(\overline{X}-\frac{S}{\sqrt{n}}t_{1-\alpha/2}(n-1),\overline{X}+\frac{S}{\sqrt{n}}t_{1-\alpha/2}(n-1)\right)$$

其中 $t_p(n)$为自由度为 n 的 t 分布的下侧 p 分位数。

同理可求得 μ 的置信度为 $1-\alpha$ 的置信上限为 $\overline{X}+\frac{S}{\sqrt{n}}t_{1-\alpha}(n-1)$

μ 的置信度为 $1-\alpha$ 的置信下限为 $\overline{X}-\frac{S}{\sqrt{n}}t_{1-\alpha}(n-1)$

$$S=\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i-\overline{X})^2}$$

例 7-4 某糖厂用自动包装机装糖，设各包重量服从正态分布 $N(\mu,\sigma^2)$。某日开工后测得 9 包重量为(单位：kg)：99.3，98.7，100.5，101.2，98.3，99.7，99.5，102.1，100.5。试求 μ 的置信度为 95%的置信区间。

解：置信度 $1-\mu=0.95$，查表得 $t_{1-\alpha/2}(n-1)=t_{0.025}(8)=2.306$。由样本值算 $\bar{x}=99.978$，$s^2=1.47$，故

置信下限 $\bar{x}-t_{1-\alpha/2}(n-1)\frac{s}{\sqrt{n}}=99.978-2.306\times\sqrt{\frac{1.47}{9}}=99.046$

置信上限 $\bar{x}+t_{1-\alpha/2}(n-1)\frac{s}{\sqrt{n}}=99.978+2.306\times\sqrt{\frac{1.47}{9}}=100.91$

所以 μ 的置信度为 95%的置信区间为(99.046，100.91)。

为此，我们编制 Python 程序如下：

```
import numpy as np
import scipy.stats as ss
from scipy.stats import t
n = 9; p = 0.025; s = np.sqrt(1.47)
x = [99.3,98.7,100.5,101.2,98.3,99.7,99.5,102.1,100.5]
xbar = np.mean(x)
low = xbar - ss.t.ppf(1 - p,n - 1) * (s / np.sqrt(n))
up = xbar + ss.t.ppf(1 - p,n - 1) * (s / np.sqrt(n))
print 'low = ',low
print 'up = ',up
```

得到如下结果：

```
low = 99.0458173021
up = 100.909738253
```

7.4 单正态总体方差区间估计的 Python 应用

此时虽然也可以就均值是否已知分两种情况讨论方差的区间估计，但在实际中 μ 已知的情形是极为罕见的，所以只在 μ 未知的条件下讨论方差 σ^2 的置信区间。

由于 $$\chi^2=(n-1)S^2/\sigma^2\sim\chi^2(n-1)$$

所以由 $$P\left(\chi^2_{\alpha/2}(n-1)<\frac{(n-1)S^2}{\sigma^2}<\chi^2_{1-\alpha/2}(n-1)\right)=1-\alpha$$

就可以得出 σ^2 的置信水平为 $1-\alpha$ 的置信区间：

$$\left(\frac{(n-1)S^2}{\chi^2_{\alpha/2}(n-1)},\frac{(n-1)S^2}{\chi^2_{1-\alpha/2}(n-1)}\right)$$

例 7-5 从某车间加工的同类零件中抽取 16 件，测得零件的平均长度为 12.8 厘米，方差为 0.0023。假设零件的长度服从正态分布，试求总体方差及标准差的置信区间(置信度为 95%)。

解：已知：$n=16,S^2=0.0023,1-\alpha=0.95$，查表得

$$\chi^2_{1-\alpha/2}(n-1)=\chi^2_{0.975}(15)=6.262$$

$$\chi^2_{\alpha/2}(n-1)=\chi^2_{0.025}(15)=27.488$$

代入数据，可算得所求的总体方差的置信区间为(0.0013，0.0055)，总体标准差的置信区间为(0.0354，0.0742)。

为此，我们编制 Python 程序如下：

```
from scipy.stats import chi2
n = 16;sq = 0.0023;p = 0.025
low = ((n - 1) * sq)/ chi2.ppf(1 - p, n - 1)
up = ((n - 1) * sq)/ chi2.ppf(p, n - 1)
print 'low = ',low
print 'up = ',up
```

得到如下结果：

```
low = 0.00125507519379
up = 0.00550930067801
```

由运行显示可知总体方差的区间估计为(0.00125507519379，0.00550930067801)。

7.5 双正态总体均值差区间估计的 Python 应用

本节讨论两个正态总体均值差的区间估计问题。

7.5.1　两方差已知时两均值差的置信区间

假设 σ_1^2,σ_2^2 都已知，要求 $\mu_1-\mu_2$ 置信水平为 $1-\alpha$ 的置信区间。

由于
$$\bar{X}\sim N(\mu_1,\sigma_1^2),\bar{Y}\sim N(\mu_2,\sigma_2^2)$$
且两者独立，得到
$$\bar{X}-\bar{Y}\sim N(\mu_1-\mu_2,\sigma_1^2/n_1+\sigma_2^2/n_2)$$
因此有
$$Z=\frac{(\bar{X}-\bar{Y}-(\mu_1-\mu_2))}{\sqrt{\sigma_1^2/n_1+\sigma_2^2/n_2}}\sim N(0,1)$$
由 $P(-z_{1-\alpha/2}<Z<z_{1-\alpha/2})=1-\alpha$ 即得
$$P\left(\bar{X}-\bar{Y}-z_{1-\alpha/2}\sqrt{\sigma_1^2/n_1+\sigma_2^2/n_2}<\mu_1-\mu_2<\bar{X}-\bar{Y}+z_{1-\alpha/2}\sqrt{\sigma_1^2/n_1+\sigma_2^2/n_2}\right)=1-\alpha$$
所以两均值差的置信区间为
$$\left(\bar{X}-\bar{Y}-z_{1-\alpha/2}\sqrt{\sigma_1^2/n_1+\sigma_2^2/n_2},\bar{X}-\bar{Y}+z_{1-\alpha/2}\sqrt{\sigma_1^2/n_1+\sigma_2^2/n_2}\right)$$
同理可求得两均值差的置信度为 $1-\alpha$ 的置信上限为
$$\bar{X}-\bar{Y}+z_{1-\alpha}\sqrt{\sigma_1^2/n_1+\sigma_2^2/n_2}$$
两均值差的置信度为 $1-\alpha$ 的置信下限为
$$\bar{X}-\bar{Y}-z_{1-\alpha}\sqrt{\sigma_1^2/n_1+\sigma_2^2/n_2}$$
下面看一个例子。

例 7-6　为比较两种农产品的产量，选择 18 块条件相似的试验田，采用相同的耕作方法做实验，结果播种甲品种的 8 块试验田的单位面积产量和播种乙品种的 10 块试验田的单位面积产量分别如表 7-2 所示。

甲品种	628，583，510，554，612，523，530，615
乙品种	535，433，398，470，567，480，498，560，503，426

假定每个品种的单位面积产量均服从正态分布，甲品种产量的方差为 2140，乙品种产量的方差为 3250。试求这两个品种平均面积产量差的置信区间(取 $\alpha=0.05$)。

为此，我们编制 Python 程序如下：

```
import numpy as np
import scipy.stats as ss
x = [628,583,510,554,612,523,530,615]
y = [535,433,398,470,567,480,498,560,503,426]
n1 = len(x);n2 = len(y)
xbar = np.mean(x);ybar = np.mean(y)
sigmaq1 = 2140;sigmaq2 = 3250;p = 0.025
low = xbar - ybar - ss.norm.ppf(q = 1 - p) * np.sqrt(sigmaq1/n1 + sigmaq2/n2)
up = xbar - ybar + ss.norm.ppf(q = 1 - p) * np.sqrt(sigmaq1/n1 + sigmaq2/n2)
print 'low = ',low
print 'up = ',up
```

得到如下结果：

```
low = 34.6870180564
up = 130.062981944
```

7.5.2 两方差都未知时两均值的置信区间

设两方差均未知，但 $\sigma_1^2=\sigma_2^2=\sigma^2$，此时由于

$$Z=\frac{\overline{X}-\overline{Y}-(\mu_1-\mu_2)}{\sqrt{\sigma_1^2/n_1+\sigma_2^2/n_2}}\sim N(0,1)$$

$$\frac{(n_1-1)S_1^2}{\sigma^2}\sim\chi^2(n_1-1),\quad \frac{(n_2-1)S_2^2}{\sigma^2}\sim\chi^2(n_2-1)$$

所以
$$\frac{(n_1-1)S_1^2}{\sigma^2}+\frac{(n_2-1)S_2^2}{\sigma^2}\sim\chi^2(n_1+n_2-2)$$

由此可得

$$T=\frac{\overline{X}-\overline{Y}-(\mu_1-\mu_2)}{\sqrt{(1/n_1+1/n_2)S^2}}\sim t(n_1-n_2-2)$$

其中，$S^2=\dfrac{(n_1-1)S_1^2+(n_2-1)S_2^2}{(n_1-1)+(n_2-1)}$。

同样由
$$P(-t_{1-\alpha/2}(n_1+n_2-2)<T<t_{1-\alpha/2}(n_1+n_2-2))=1-\alpha$$

解不等式即得两均值差的置信度为 $1-\alpha$ 的置信区间：

$$(\overline{X}-\overline{Y}\pm t_{1-\alpha/2}(n_1+n_2-2)\sqrt{(1/n_1+1/n_2)S^2})$$

同理可求得两均值差的置信度为 $1-\alpha$ 的置信上限为

$$(\overline{X}-\overline{Y}+t_{1-\alpha/2}(n_1+n_2-2)\sqrt{(1/n_1+1/n_2)S^2})$$

两均值差的置信度为 $1-\alpha$ 的置信下限为

$$(\overline{X}-\overline{Y}-t_{1-\alpha/2}(n_1+n_2-2)\sqrt{(1/n_1+1/n_2)S^2})$$

例 7-7 在上一例题中，如果不知道两种品种产量的方差，但已知两者相同，求置信区间。

为此，我们编制 Python 程序如下：

```
import numpy as np
import scipy.stats as ss
x = [628,583,510,554,612,523,530,615]
y = [535,433,398,470,567,480,498,560,503,426]
n1 = 1.0 * len(x);n2 = 1.0 * len(y) # 转为小数
s1 = np.var(x);s2 = np.var(y)
xbar = np.mean(x);ybar = np.mean(y)
p = 0.025
sq = ((n1 - 1) * s1 + (n2 - 1) * s2)/(n1 - 1 + n2 - 1)
low = xbar - ybar - ss.t.ppf(1 - p,n1 + n2 - 2) * np.sqrt(sq * (1/n1 + 1/n2))
up = xbar - ybar + ss.t.ppf(1 - p,n1 + n2 - 2) * np.sqrt(sq * (1/n1 + 1/n2))
print 'low = ',low
print 'up = ',up
```

得到如下结果：

```
low = 32.4209278184
up = 132.329072182
```

可见,这两个品种的单位面积产量方差的置信水平为 0.95 的置信区间为(32.4209278184,132.329072182)。

7.6　双正态总体方差比区间估计的 Python 应用

此时虽然也可以就均值是否已知分两种情况讨论方差的区间估计,但在实际中 μ 已知的情形是极为罕见的,所以只讨论在 μ 未知的条件下方差 σ^2 的置信区间。

由于　　$(n_1-1)S_1^2/\sigma^2 \sim \chi^2(n_1-1), (n_2-1)S_2^2/\sigma^2 \sim \chi^2(n_2-1)$

且 S_1^2 与 S_2^2 相互独立,故

$$F = (S_1^2/\sigma_1^2)/(S_2^2/\sigma_2^2) \sim F(n_1-1, n_2-1)$$

所以对于给定的置信水平 $1-\alpha$,由

$$P(F_{\alpha/2}(n_1-1, n_2-1) < (S_1^2/\sigma_1^2)/(S_2^2/\sigma_2^2) < F_{1-\alpha/2}(n_1-1, n_2-1)) = 1-\alpha$$

就可以得出两方差比的置信水平为 $1-\alpha$ 的置信区间

$$\left(\frac{S_1^2}{S_2^2}\frac{1}{F_{1-\alpha/2}(n_1-1, n_2-1)}, \frac{S_1^2}{S_2^2}\frac{1}{F_{\alpha/2}(n_1-1, n_2-1)}\right)$$

其中,$F_p(m,n)$为自由度为(m,n)的 F 分布的下侧 p 分位数。

例 7-8　甲、乙两台机床分别加工某种轴承,轴承的直径分别服从正态分布 $N(\mu_1,\sigma_1^2)$,$N(\mu_2,\sigma_2^2)$,从各自加工的轴承中分别抽取若干个轴承测其直径,结果如表 7-3 所示。

总体	样本容量	直　径
X(机床甲)	8	20.5,19.8,19.7,20.4,20.1,20.0,19.0,19.9
X(机床乙)	7	20.7,19.8,19.5,20.8,20.4,19.6,20.2

试求两台机床加工的轴承直径的方差比的 0.95 的置信区间。

```
import numpy as np
from scipy.stats import f
x = [20.5,19.8,19.7,20.4,20.1,20.0,19.0,19.9]
y = [20.7,19.8,19.5,20.8,20.4,19.6,20.2]
sq1 = np.var(x);sq2 = np.var(y)
n1 = 8;n2 = 7;p = 0.025
f.ppf(0.025, n1 - 1, n2 - 1)
low  =  sq1/sq2 * 1/f.ppf(1 - p, n1 - 1, n2 - 1)
up  =  sq1/sq2 * 1/f.ppf(p, n1 - 1, n2 - 1)
print 'low = ',low
print 'up = ',up
low =  0.142168867371
up =  4.14462281408
```

由上面的运行显示可见,两台机床的加工轴承的直径的方差比的 0.95 置信区间(0.142168867371,4.14462281408),方差比为 0.7932。

练　习　题

对本章例题,使用 Python 重新操作一遍。

第8章

参数假设检验的 Python 应用

参数假设检验是指对参数的平均值、方差、比率等特征进行的统计检验。参数假设检验一般假设统计总体的具体分布是已知的，但是其中的一些参数或者取值范围不确定，分析的主要目的是估计这些未知参数的取值，或者对这些参数进行假设检验。参数假设检验不仅能够对总体的特征参数进行推断，还能够对两个或多个总体的参数进行比较。常用的参数假设检验包括单一样本 t 检验、两个总体均值差异的假设检验、总体方差的假设检验、总体比率的假设检验等。本章先介绍假设检验的基本理论，然后通过实例来说明 R 软件在参数假设检验中的具体应用。

8.1 参数假设检验的基本理论

8.1.1 假设检验的概念

为了推断总体的某些性质，我们会提出总体性质的各种假设。假设检验就是根据样本提供的信息对所提出的假设作出判断的过程。

原假设是我们有怀疑，想要拒绝的假设，记为 H_0。备择假设是我们拒绝了原假设后得到的结论，记为 H_α。

假设都是关于总体参数的，例如，我们想知道总体均值是否等于某个常数 μ_0，那么原假设是 $H_0:\mu=\mu_0$，则备择假设是 $H_\alpha:\mu\neq\mu_0$。

上面这种假设，我们称为双尾检验，因为备择假设是双边的。

下面两种假设检验称为单尾检验：

$$H_0:\mu\geqslant\mu_0\quad H_\alpha:\mu<\mu_0$$

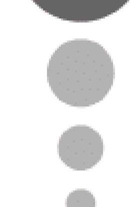
$$H_0:\mu\leqslant\mu_0\quad H_\alpha:\mu>\mu_0$$

注意：无论是单尾还是双尾检验，等号永远都在原假设一边，这是用来判断原来假设的唯一标准。

8.1.2 第一类错误和第二类错误

我们在做假设检验的时候会犯两种错误：第一，原来假设是正确的而你判断它为错误的；第二，原来假设是错误的而你判断它为正确的。我们分别称为第一类错误和第二类错误。

第一类错误：原来假设是正确的，却拒绝了原来假设；



第二类错误：原来假设是错误的，却没有拒绝原来假设。

这类似于法官判案时，如果被告是好人，却判他为坏人，这是第一类错误（错杀好人或以真为假）。

如果被告是坏人，却判他为好人，这是第二类错误（放走坏人或以假为真）。

在其他条件不变的情况下，如果要求犯第一类错误概率越小，那么犯第二类错误的概率就会越大，通俗的理解是：当我们要求错杀好人的概率降低，那么往往就会放走坏人。

同样的，在其他条件不变的情况下，如果要求犯第二类错误概率越小，那么犯第一类错误的概率就越大。通俗的理解即：当我们要求放走坏人的概率降低，那么往往就会错杀好人。

其他条件不变主要指的是样本量 n 不变。换言之，要想减少犯第一类错误的概率和犯第二类错误的概率，就要增大样本量 n。

在假设检验的时候，我们会规定一个允许犯第一类错误的概率，比如 5%，这称为显著性水平，记为 α。我们通常只规定犯第一类错误的概率，而不规定犯第二类错误的概率。

检验的势定义为在原假设是错误的情况下正确拒绝原假设的概率。检验的势等于 1 减去犯第二类错误的概率。

我们用表 8-1 来表示显著性水平和检验的势。

表 8-1　显著性水平和检验的势

项　　目	原假设正确	原假设不正确
拒绝原假设	第一类错误 显著性水平 α	判断正确 检验的势＝1－P(第二类错误)
没有拒绝原假设	判断正确	第二类错误

要做假设检验，我们先要计算两样东西：检验统计量和关键值。

检验统计量是从样本数据中计算得来的。检验统计量的一般形式为：

检验统计量＝(样本统计量－在 H_0 中假设的总体参数值)/样本统计量的标准误

关键值是查表得到的。关键值的计算需要知道以下三点。

(1) 检验统计量是什么分布，这决定我们要去查哪张表；

(2) 显著性水平；

(3) 是双尾还是单尾检验。

8.1.3　决策规则

1. 基于检验统计量和关键值的决策准则

计算检验统计量和关键值之后，怎样判断是拒绝原假设还是不拒绝原假设呢?

首先，我们要搞清楚我们做的是双尾检验还是单尾检验。如果是双尾检验，那么拒绝域在两边。以双尾 Z 检验为例，首先画出 Z 分布（标准正态分布），在两边画出黑色的拒绝区域。如图所示。

拒绝区域的面积应等于显著性水平。以 $\alpha=0.05$ 为例，左右两块拒绝区域的面积之和应等于 0.05，可知交界处的数值为±1.96。±1.96 即为关键值，如图 8-1 所示。

如果从样本数据中计算得出的检验统计量落在拒绝区域（小于－1.96 或大于 1.96），就

拒绝原假设；如果检验统计量没有落在拒绝区域（在−1.96 和 1.96 之间），就不能拒绝原假设。

如果是单尾检验，那么拒绝区域在一边。拒绝区域在哪一边，要看备择假设在哪一边。以单尾的 Z 检验为例，假设原假设为 $H_0: \mu \leqslant \mu_0$，备择假设为 $H_a: \mu > \mu_0$，那么拒绝区域在右边，因为备择假设在右边。首先画出 z 分布（标准正态分布），在右边画出黑色的拒绝区域，如图 8-2 所示。

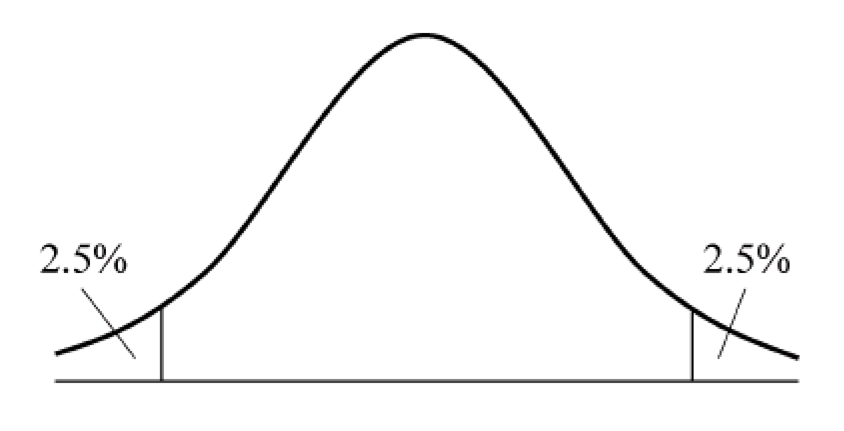


图 8-1　双边拒绝域的正态分布图

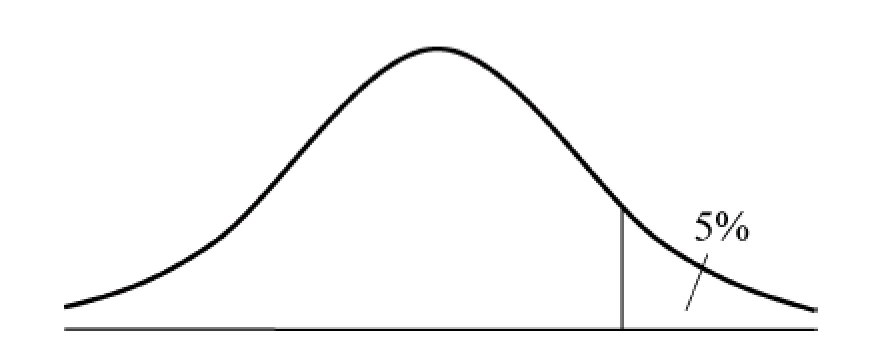


图 8-2　右边拒绝域的正态分布图

拒绝区域的面积还是等于显著性水平。以 $\alpha=0.05$ 为例，因为只有一块拒绝区域，因此其面积为 0.05，可知交界处的数值为 1.65。1.65 即为关键值。

如果从样本数据中计算得出的检验统计量落在拒绝区域（大于 1.65），就拒绝原假设；如果检验统计量没有落在拒绝区域（小于 1.65），就不能拒绝原假设。

2. 基于 p 值和显著性水平的决策规则

在实际中，如统计软件经常给出是 p 值，可以将 p 值与显著性水平作比较，以决定拒绝还是不拒绝原假设，这是基于 p 值和显著性水平的决策规则。

首先来看看 p 值到底是什么。对于双尾检验，有两个检验统计量，两个统计量两边的面积之和就是 p 值。因此，每一边的面积是 $p/2$，如图 8-3 所示。

对于单尾检验，只有一个检验统计量，检验统计量边上的面积就是 p 值，如图 8-4 所示。

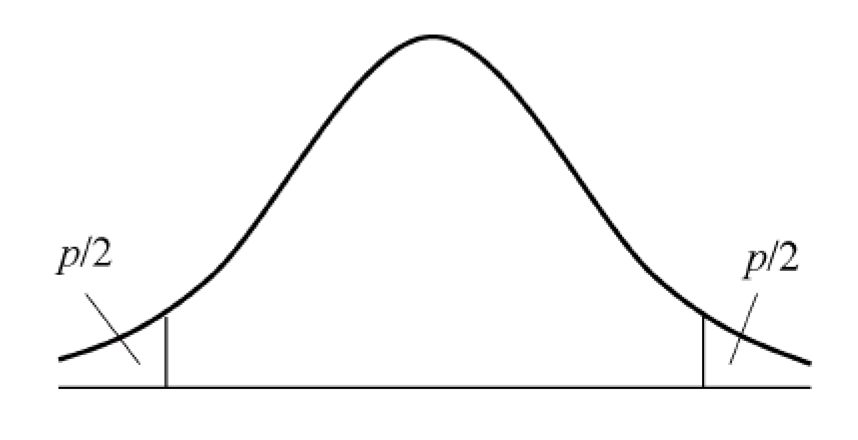


图 8-3　双边 p 值的正态分布图

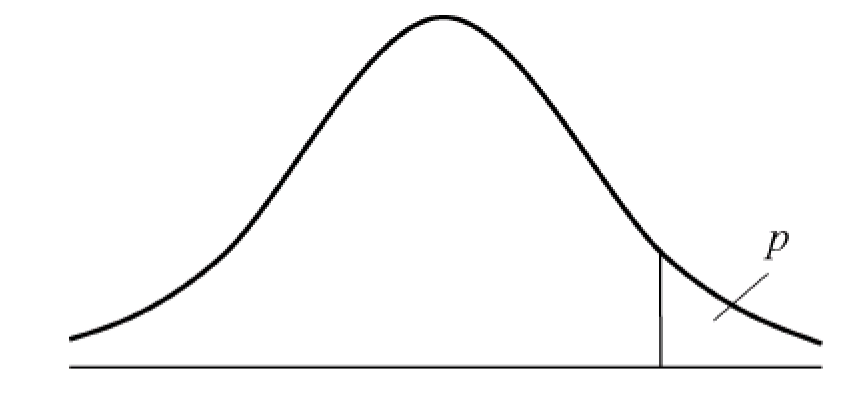


图 8-4　单边 p 值的正态分布图

计算 p 值的目的是与显著性水平作比较。如果 p 值小于显著性水平，说明检验统计量落在拒绝区域，因此拒绝原假设。如果 p 值大于显著性水平，说明检验统计量没有落在拒绝区域，因此不能拒绝原假设。

p 值的定义为：可以拒绝原假设的最小显著性水平。

3. 结论的陈述

如果不能拒绝原假设，我们不能说接受原假设，只能说“不能拒绝原假设”（can not

reject H_0 或 fail to reject H_0)。

在作出判断之后,我们还要陈述结论。如果拒绝原假设,那么我们说总体均值显著地不相等。

8.1.4 单个总体均值的假设检验

我们想知道一个总体均值是否等于(或大于等于、小于等于)某个常数 μ_0,可以使用 Z 检验或 T 检验。双尾和单尾检验的原假设和备择假设如下:

$$H_0:\mu=\mu_0,\quad H_a:\mu\neq\mu_0$$
$$H_0:\mu\geqslant\mu_0,\quad H_a:\mu<\mu_0$$
$$H_0:\mu\leqslant\mu_0,\quad H_a:\mu>\mu_0$$

下表 8-2 告诉我们什么时候使用 Z 检验,什么时候使用 t 检验。

表 8-2 Z 检验与 t 检验比较

	正态总体,$n<30$	$n>=30$
已知总体方差	Z 检验	Z 检验
未知总体方差	t 检验	t 检验或 Z 检验

下面,我们要计算 Z 统计量和 t 统计量。

如果已知总体方差,那么 Z 统计量的公式为

$$z=\frac{\bar{x}-\mu_0}{\sigma\sqrt{n}}$$

其中,$\bar{x}$ 为样本均值,σ 为总体标准差,n 为样本容量。

如果未知总体方差,那么 Z 统计量的公式为

$$z=\frac{\bar{x}-\mu_0}{s\sqrt{n}}$$

其中,$\bar{x}$ 为样本均值,s 为样本标准差,$\left(n>30, s^2=\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2, n<30, s^2=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2\right)$,$n$ 为样本容量。

t 统计量的公式为

$$t_{n-1}=\frac{\bar{x}-\mu_0}{s\sqrt{n}}$$

其中 $\bar{x}$ 为样本均值,s 为样本标准差,n 为样本容量。

下标 $n-1$ 是 t 分布的自由度,我们在查表找关键值时要用到自由度。

例 8-1 一个股票型共同基金的风险收益特征。一家已经在市场中生存了 24 个月的中等市值成长型基金。在这个区间中,该基金实现了 1.50%的月度平均收益率,而且该月度收益率的样本标准差为 3.6%。给定该基金所面临的系统性风险(市场风险)水平,并根据一个定价模型,我们预期该共同基金在这个区间中应该获得 1.10%的月度平均收益率。假定收益率服从正态分布,那么实际结果是否和 1.1%这个理论上的月度平均收益率或者总体月度平均收益率相一致?

(1) 给出与该研究项目的语言描述相一致的原假设和备择假设;

(2) 找出对于第(1)问中的假设进行检验的检验统计量;

(3) 求出 0.10 显著性水平下第 1 问中所检验的假设的拒绝点;

(4) 确定是否应该在 0.10 显著性水平下拒绝原假设。

解:(1) 我们有一个“不等”的备择假设,其中 μ 是该股票基金的对应的平均收益率 $H_0:\mu=1.1$ 对应于 $H_a:\mu\neq1.1$。

(2) 因为总体方差是未知的,我们利用 24−1=23 自由度的 t 检验。

(3) 因为这是一个双边检验,我们的拒绝点 $t_{n-1}=t_{0.05,23}$,在 t 分布表中,自由度为 23 的行和 0.05 的列,找到 1.714。双边检验的两个拒绝点是 1.714 和 −1.714。如果我们发现 $t>1.714$ 或 $t<1.714$,我们将拒绝原假设。

(4) $t_{23}=\dfrac{\bar{x}-\mu_0}{s\sqrt{n}}=\dfrac{1.50-1.10}{3.6\%/\sqrt{24}}=0.544331$ 或 0.544。

8.1.5 两个独立总体均值的假设检验

我们想知道两个相互独立的正态分布总体的均值是否相等,可以使用 t 检验来完成。双尾和单尾检验的原假设和备择假设如下:

$$H_0:\mu_1=\mu_2,\quad H_a:\mu_1\neq\mu_2$$
$$H_0:\mu_1\geqslant\mu_2,\quad H_a:\mu_1<\mu_2$$
$$H_0:\mu_1\leqslant\mu_2,\quad H_a:\mu_1>\mu_2$$

下标 1 和 2 分别表示取自第一个总体的样本和取自第二个总体的样本,这两个样本是相互独立的。

在开始做假设检验之前,我们先要区分两种情况:第一种,两总体方差未知但假定相等;第二种,两总体方差未知且假定不等。

对于第一种情况,我们用 t 检验,其自由度为 n_1+n_2-2。t 统计量的计算公式如下:

$$t_{n_1+n_2-2}=\frac{(\bar{x}_1-\bar{x}_2)-(\mu_1-\mu_2)}{\sqrt{\dfrac{s_p^2}{n_1}+\dfrac{s_p^2}{n_2}}},\text{其中 } s_p^2=\frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-2}$$

s_1^2 为第一个样本的样本方差,s_2^2 为第二个样本的样本方差,n_1 为第一个样本的样本量,n_2 为第二个样本的样本量。

例 8-2 20 世纪 80 年代的标准普尔 500 指数已实现的月度平均收益率似乎与 20 世纪 70 年代的月度平均收益率有着巨大的不同,那么这个不同是否存在统计上是显著的呢?表 8-3 所给的数据表明,我们没有充足的理由拒绝这两个 10 年的收益率的总体方差是相同的。

表 8-3 两个 10 年的标准普尔 500 指数的月度平均收益率及其标准差

10 年区间	月数 n	月度平均收益率	标准差
20 世纪 70 年代	120	0.580	4.598
20 世纪 80 年代	120	1.470	4.738

(1) 给出与双边假设检验相一致的原假设和备择假设。

(2) 找出检验第(1)问中假设的检验统计量。

（3）求出第（1）问中所检验的假设在 0.10，0.05，0.01 显著性水平下的拒绝点。

（4）确定在 0.10，0.05 和 0.01 显著性水平下是否应拒绝原假设。

解：（1）令 μ_1 表示 20 世纪 70 年代的总体平均收益率，令 μ_2 表示 20 世纪 80 年代的总体平均收益率，于是我们给出如下的假设：

$$H_0: \mu_1 = \mu_2, \quad H_a: \mu_1 \neq \mu_2$$

（2）因为两个样本分别取自不同的 10 年区间，所以它们是独立样本。总体方差是未知的，但是可以被假设为相等。给定所有这些条件，在 t 统计量的计算公式中所给出的 t 检验具有 120＋120－2＝238 的自由度。

（3）在 t 分布表中，最接近 238 的自由度为 200。对于一个双边检验，df＝200 的 0.10，0.05，0.01 显著性水平下的拒绝点分别为±1.653，±1.972，±2.601。即在 0.10 显著性水平下，如果 $t<-1.653$ 或者 $t>1.653$，我们将拒绝原假设；在 0.05 显著性水平下，如果 $t<-1.972$ 或者 $t>1.972$，我们将拒绝原假设；在 0.01 显著性水平下，如果 $t<-2.601$ 或者 $t>2.601$，我们将拒绝原假设。

（4）计算检验统计量时，首先计算合并方差的估计值：

$$s_p^2 = \frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-2} = \frac{(120-1)(4.598)^2+(120-1)(4.738)^2}{120+120-2} = 21.795124$$

$$t_{n_1+n_2-2} = \frac{(\bar{x}_1-\bar{x}_2)-(\mu_1-\mu_2)}{\sqrt{\dfrac{s_p^2}{n_1}+\dfrac{s_p^2}{n_2}}} = \frac{(0.580-1.470)-0}{\left(\dfrac{21.795124}{120}+\dfrac{21.795124}{120}\right)^{1/2}} = \frac{-0.89}{0.602704} = -1.477$$

t 值等于－1.477 在 0.10 显著性水平下不显著，同样在 0.05 和 0.01 显著性水平下也不显著。因此，我们无法在任一个显著性水平下拒绝原假设。

当我们能假设两个总体服从正态分布，但是不知道总体方差，而且不能假设方差是相等的时候，基于独立随机样本的近似。检验给出如下：

$$t = \frac{(\bar{x}_1-\bar{x}_2)-(\mu_1-\mu_2)}{\sqrt{\dfrac{s_1^2}{n_1}+\dfrac{s_2^2}{n_2}}}$$

s_1^2 为第一个样本的样本方差，s_2^2 为第二个样本的样本方差，n_1 为第一个样本的样本量，n_2 为第二个样本的样本量。

其中，我们使用“修正的”自由度，其计算公式为

$$\mathrm{df} = \frac{(s_1^2/n_1+s_2^2/n_2)^2}{(s_1^2/n_1)^2/n_1+(s_2^2/n_2)^2/n_2}$$

的数值表。

例 8-3：违约债券的回收率。具有风险的公司债券的要求收益率是如何决定的？两个重要的考虑因素为预期违约概率和在违约发生的情况下预期能够回收的金额（即回收率）。奥特曼（Altman）和基肖尔（Kishore）（1996）首次记录了行业和信用等级进行分层的违约债券的平均回收率。对于他们的研究区间 1971—1995 年，奥特曼（Altman）和基肖尔（Kishore）发现公共事业公司、化工类公司、石油公司以及塑胶制造公司的违约债券的回收率明显要高于其他行业。这一差别是否能够通过在高回收率行业中的高信用债券比较来解释？他们通过检验以信用等级分层的回收率来对此进行研究。这里，我们仅讨论他们对于高信用担保债券的结果。其中 μ_1 表示公共事业公司的高信用担保债券的总体平均回收率，而 μ_2 表示其他行业（非公共事业）公司的高信用担保债券的总体平均回收率，假设 $H_0: \mu_1 =$

μ_2，H_a：$\mu_1 \neq \mu_2$。

表 8-4 摘自他们的部分结果。

表 8-4 高信用债券的回收率 单位：美元

行业类/高信用	非公共事业样本			非公共事业样本		
	观测数	违约时的平均价格	标准差	观测数	违约时的平均价格	标准差
公共事业高信用担保	21	64.42	14.03	64	55.75	25.17

根据他们的研究假设，总体服从正态分布，并且样本是独立的。根据表 8-4 中的数据，回答下列问题：

(1) 讨论为什么奥特曼(Altman)和基肖尔(Kishore)会选择 $t=\frac{(\bar{x}_1-\bar{x}_2)-(\mu_1-\mu_2)}{\sqrt{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}}$，而不是 $t_{n_1+n_2-2}=\frac{(\bar{x}_1-\bar{x}_2)-(\mu_1-\mu_2)}{\sqrt{\frac{s_p^2}{n_1}+\frac{s_p^2}{n_2}}}$ 的检验方法。

(2) 计算检验上述给出的原假设的检验统计量。

(3) 该检验的修正自由度的数值为多少？

(4) 确定在 0.10 显著性水平下是否应该拒绝原假设。

解：(1) 高信用担保的公共事业公司回收率的样本标准差 14.03 要比与之相比的非公共事业公司回收率的样本标准差 25.17 更小。故不假设它们的均值相等的选择是恰当的，所以奥特曼和基肖尔采用 $t=\frac{(\bar{x}_1-\bar{x}_2)-(\mu_1-\mu_2)}{\sqrt{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}}$ 检验。

(2) 检验统计量为

$$t=\frac{(\bar{x}_1-\bar{x}_2)-(\mu_1-\mu_2)}{\sqrt{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}}$$

式中，$\bar{x}_1$ 表示公共事业公司的样本平均回收率 $=64.42$，$\bar{x}_2$ 表示非公共事业公司的样本平均回收率 $=55.75$，$s_1^2=14.03$ ^2 $=196.8409$，$s_2^2=25.17$ ^2 $=633.5289$，$n_1=21$，$n_2=64$。

因此

$$t=\frac{(\bar{x}_1-\bar{x}_2)-(\mu_1-\mu_2)}{\sqrt{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}}=\frac{64.42-5575}{[196.8409/21+633.5289/64]^{1/2}}=1.975$$

(3)

$$\text{df}=\frac{(s_1^2/n_1+s_2^2/n_2)^2}{(s_1^2/n_1)^2/n_1+(s_2^2/n_2)^2/n_2}$$

$$=\frac{(196.8409/21+633.5289/64)^2}{(196.8409/21)^2/21+(633.5289/64)^2/64}=64.99$$

即 65 个自由度。

(4) 在 t 分布表的数值表中最接近 df=65 的一栏是 df=60。对于 $\alpha=0.10$，我们找到 $t_{\alpha/2}=1.671$。因此，如果 $t<-1.671$ 或 $t>1.671$，我们就会拒绝原假设。基于所计算的值 $t=1.975$，我们在 0.10 显著性水平下拒绝原假设。存在一些公共事业公司和非公共事业公司回收率不同的证据。为什么是这样的？奥特曼(Altman)和基肖尔(Kishore)认为公司资

产的不同性质以及不同行业的竞争水平造成了不同的回收率。

8.1.6　成对比较检验

上面我们讲的是两个相互独立的正态分布总体的均值检验，两个样本是相互独立的。如果两个样本相互不独立，我们做均值检验时要使用成对比较检验。成对比较检验也使用 t 检验来完成，双尾和单尾检验的原假设和备择假设如下：

$$H_0: \mu_d = \mu_0, \quad H_a: \mu_d \neq \mu_0$$
$$H_0: \mu_d \geqslant \mu_0, \quad H_a: \mu_d < \mu_0$$
$$H_0: \mu_d \leqslant \mu_0, \quad H_a: \mu_d > \mu_0$$

其中 μ_d 表示两个样本均值之差，为常数，μ_0 通常等于 0。

t 统计量的自由度为 $n-1$，计算公式如下：

$$t = \frac{\bar{d} - \mu_0}{s_{\bar{d}}}$$

其中，$\bar{d}$是样本差的均值。我们取得两个成对的样本之后，对应相减，就得到一组样本差的数据，求这一组数据的均值，就是$\bar{d}$。$s_{\bar{d}}$ 是$\bar{d}$的标准误，即 $s_{\bar{d}} = s_d/\sqrt{n}$。

下面的例子说明了对于竞争的投资策略进行评估的这个检验的应用。

例 8-5　道-10 投资策略。麦奎因（Mcqueen）、谢尔德斯（Shields）和索利（Thorley）（1997）检验了一个流行的投资策略（该策略投资于道琼斯工业平均指数中收益率最高的 10 只股票）与一个买入并持有的策略（该策略投资于道琼斯工业平均指数中所有的 30 只股票）之间的业绩比较。他们研究的区间段是 1946—1995 年的 50 年区间。

表 8-5　道-10 和道-30 投资组合年度收益率汇总（1946—1995）（n=50）

策略	平均收益率	标准差
道-10	16.77%	19.10%
道-30	13.71	16.64
差别	3.06	6.62①

注：① 差别的样本标准差。

请回答下列问题：

（1）给出道-10 和道-30 策略间收益率差别的均值等于 0 这个双边检验相一致的原假设和备择假设。

（2）找出对于第（1）问中假设进行检验的检验统计量。

（3）求出在 0.01 显著性水平下第（1）问中所检验的假设的拒绝点。

（4）确定在 0.01 显著性水平下是否应该拒绝原假设。

（5）讨论为什么选择配对比较检验。

解：（1）μ_d 表示道-10 和道-30 策略间收益率差别的均值，我们有

$H_0: \mu_d = 0$，$H_a: \mu_d \neq 0$。

（2）因为总体方差未知，所有检验统计量为一个自由度 $50-1=49$ 的 t 检验。

（3）在 t 分布表中，我们查阅自由度为 49 的一行，显著性水平为 0.05 的一列，从而得到 2.68。如果我们发现 $t>2.68$ 或 $t<-2.68$。我们将拒绝原假设。

(4) $t=\frac{3.06}{6.62/\sqrt{50}}=3.2685$ 或 3.27。

因为 3.27>2.68,所以我们拒绝原假设。

结论:平均收益率的差别在统计上是明显显著的。

(5) 道-30 包含道-10。因此,它们不是相互独立的样本;通常,道-10 和道-30 策略间收益率的相关系数为正。因为样本是相互依赖的,配对比较检验是恰当的。

8.1.7 单个总体方差的假设检验

首先是关于单个总体方差是否等于(大于等于、小于等于)某个常数的假设检验。我们要使用卡方检验。

双尾和单尾检验的原假设和备择假设如下:

$$H_0:\sigma^2=\sigma_0^2,\quad H_a:\sigma^2\neq\sigma_0^2$$
$$H_0:\sigma^2\geqslant\sigma_0^2,\quad H_a:\sigma^2<\sigma_0^2$$
$$H_0:\sigma^2\leqslant\sigma_0^2,\quad H_a:\sigma^2>\sigma_0^2$$

卡方统计量的自由度为 $n-1$,计算方法如下:

$$\chi^2=\frac{(n-1)s^2}{\sigma_0^2}$$

其中 s^2 为样本方差。

例如:某股票的历史月收益率的标准差为 5%,这一数据是基于 2003 年以前的历史数据测定的。现在,我们选取 2004—2006 年这 36 个月的月收益率数据,来检验其标准差是否还为 5%。我们测得这 36 月的月收益率标准差为 6%。以显著性水平为 0.05,检验其标准差是否还为 5%,结果如何?

(1) 写出原假设和备择假设

$$H_0:\sigma^2=(5\%)^2,\quad H_a:\sigma^2\neq(5\%)^2$$

(2) 使用卡方检验

(3) $\chi^2=\frac{(n-1)s^2}{\sigma_0^2}=(36-1)\times(6\%)/(5\%)^2=50.4$

(4) 查表得到卡方关键值。对于显著性水平 0.05,由于是双尾检验,两边的拒绝区域面积都为 0.025,自由度为 35,因此关键值为 20.569 和 53.203。

(5) 由于 50.4<53.203,卡方统计量没有落在拒绝区域,因此我们不能拒绝原假设。

(6) 最后我们陈述结论:该股票的标准差没有显著地不等于 5%。

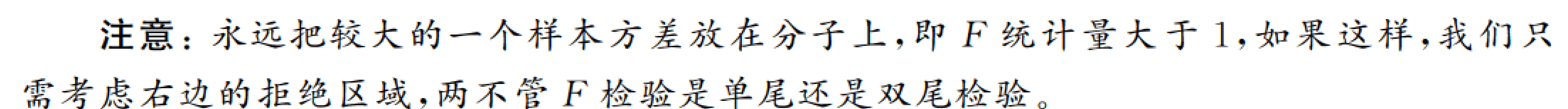
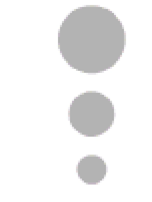
8.1.8 两个总体方差的假设检验

双尾和单尾检验的原假设和备择假设如下。

$$H_0:\sigma_1^2=\sigma_2^2,\quad H_a:\sigma_1^2\neq\sigma_2^2$$
$$H_0:\sigma_1^2\geqslant\sigma_2^2,\quad H_a:\sigma_1^2<\sigma_2^2$$
$$H_0:\sigma_1^2\leqslant\sigma_2^2,\quad H_a:\sigma_1^2>\sigma_2^2$$

F 统计量的自由度为 n_1-1 和 n_2-1。

$$F=s_1^2/s_2^2$$

注意：永远把较大的一个样本方差放在分子上，即 F 统计量大于 1，如果这样，我们只需考虑右边的拒绝区域，两不管 F 检验是单尾还是双尾检验。

例 8-6　我们想检验 IBM 股票和 HP 股票的月收益率的标准差是否相等。我们选取 2004—2006 年这 36 个月的月收益率数据，来检验其标准差是否还为 5%。我们测得这 36 个月它们的月收益率标准差分别为 5% 和 6%。以显著性水平为 0.05，假设检验的结果如何？

（1）写出原假设和备择假设

$$H_0: \sigma_1^2 = \sigma_2^2, \quad H_a: \sigma_1^2 \neq \sigma_2^2$$

（2）使用 F 检验。

（3）计算 F 统计量 $F = s_1^2/s_2^2 = 0.0036/0.0025 = 1.44$。

（4）查表得到 F 关键值 2.07。

（5）由于 $1.44 < 2.07$，F 统计量没有落在拒绝区域，因此我们不能拒绝原假设。

（6）最后我们陈述结论：IBM 股票和 HP 股票的标准差没有显著地不等。

8.2　单个样本 t 检验的 Python 应用

单个样本 t 检验是假设检验中最基本也是最常用的方法之一。与所有的假设检验一样，其依据的基本原理也是统计学中的“小概率反证法”原理。通过单个样本 t 检验，可以实现样本均值和总体均值的比较。检验的基本步骤是：首先提出原假设和备择假设，规定好检验的显著性水平，然后确定适当的检验统计量，并计算检验统计量的值，最后依据计算值和临界值的比较做出统计决策。

例 8-7　某电脑公司销售经理人均月销售 500 台电脑，现采取新的广告政策，半年后，随机抽取该公司 20 名销售经理的人均月销售量数据，具体数据如表 8-6 所示。问：广告策略是否能够影响销售经理的人均月销售量？

表 8-6　人均月销售量　　单位：台

编号	人均月销售量	编号	人均月销售量
1	506	11	510
2	503	12	504
3	489	13	512
4	501	14	499
5	498	15	487
6	497	16	507
7	491	17	503
8	502	18	488
9	490	19	521
10	511	20	517

在目录 G:\2glkx\data 下建立 al8-1.xls 数据文件后，使用如下命令取数。

```
import pandas as pd
import numpy as np
```

```
#读取数据并创建数据表,名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al8 - 1.xls '))
#查看数据表前 5 行的内容
data.head()
```

得到前 5 条记录的数据如下：

```
   sale
0   506
1   503
2   489
3   501
4   498
#取 sale 数据
x = np.array(data[['sale']])
mu = np.mean(x)
from scipy import stats as ss
print mu,ss.ttest_1samp(a = x,popmean = 500)
mu 501.8 Ttest_1sampResult(statistic = array([ 0.83092969]), pvalue = array([ 0.41633356]))
```

通过观察上面的分析结果，可以看出样本均值是 501.8，样本的 t 值为 0.83092969，p 值为 0.41633356，远大于 0.05，因此不能拒绝原假设，也就是说，广告策略不能影响销售经理的人均月销售量。

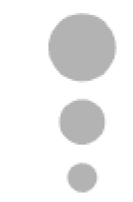
8.3 两个独立样本 t 检验的 Python 应用

Python 的独立样本 t 检验是假设检验中最基本也是最常用的方法之一。与所有的假设检验一样，其依据的基本原理也是统计学中的“小概率反证法”原理。通过独立样本 t 检验，可以实现两个独立样本的均值比较。两个独立样本 t 检验的基本步骤也是首先提出原假设和备择假设，规定好检验的显著性水平，然后确定适当的检验统计量，并计算检验统计量的值，最后依据计算值和临界值的比较作出统计决策。

例 8-8 表 8-7 给出了 a、b 两个基金公司各管理 40 只基金的价格。试用独立样本 t 检验方法研究两个基金公司所管理的基金价格之间有无明显的差别（设定显著性水平为 5%）。

表 8-7 a、b 两个基金公司各管理基金的价格 单位：元

编号	基金 a 价格	基金 b 价格
1	145	101
2	147	98
3	139	87
4	138	106
5	145	101
…	…	…
38	138	105
39	144	99
40	102	108



虽然这里两只基金的样本相同，但要注意的是：两个独立样本 t 检验并不需要两样本数相同。

在目录 G:\2glkx\data 下建立 al8-2.xls 数据文件后，取数的命令如下：

```
import pandas as pd
import numpy as np
#读取数据并创建数据表，名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al8 - 2.xls '))
#查看数据表前 5 行的内容
data.head()
     fa   fb
0   145   101
1   147    98
2   139    87
3   138   106
4   135   105
x  =  np.array(data[['fa']])
y  =  np.array(data[['fb']])
from scipy.stats import ttest_ind
t,p = ttest_ind(x,y)
print 't = ',t
print 'p = ',p
```

得到如下结果：

```
t =  [ 14.04978844]
p =  [ 4.54986161e - 23]
```

通过观察上面的分析结果，可以看出：t 值＝14.04978844，p 值＝4.54986161e-23，远小于 0.05，因此需要拒绝原假设。也就是说，两家基金公司被调查的基金价格之间存在明显的差别。

8.4　配对样本 t 检验的 Python 应用

Python 的配对样本 t 检验过程也是假设检验中的方法之一。与所有的假设检验一样，其依据的基本原理也是统计学中的"小概率反证法"原理。通过配对样本 t 检验，可以实现对称成对数据的样本均值比较。与独立样本 t 检验的区别是：两个样本来自于同一总体，而且数据的顺序不能调换。配对样本 t 检验的基本步骤也是首先提出原假设和备择假设，规定好检验的显著性水平，然后确定适当的检验统计量，并计算检验统计量的值，最后依据计算值和临界值的比较做出统计决策。

例 8-9　为了研究一种政策的效果，特抽取了 50 只股票进行了试验，实施政策前后股票的价格如表 8-8 所示。试用配对样本 t 检验方法判断该政策能否引起研究股票价格的明显变化(设定显著性水平为 5%)。

表 8-8　政策实施前后的股票价格　　单位：元

编号	政策前价格	政策后价格
1	88.60	75.60
2	85.20	76.50
3	75.20	68.20
…	…	…
48	82.70	78.10
49	82.40	75.30
50	75.60	69.90

在目录 G:\2glkx\data 下建立 al8-3.xls 数据文件后，取数的命令如下：

```
import pandas as pd
import numpy as np
#读取数据并创建数据表，名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al8 - 3.xls '))
#查看数据表前 5 行的内容
data.head()
      qian       hou
    0  88.599998  75.599998
    1  85.199997  76.500000
    2  75.199997  68.199997
    3  78.400002  67.199997
    4  76.000000  69.900002
    x = np.array(data[['qian']])
    y = np.array(data[['hou']])
    from scipy.stats import ttest_rel
    t,p = ttest_rel(x,y)
    print 't = ',t
    print 'p = ',p
```

得到如下结果：

```
t = [ 12.43054293]
p = [ 9.13672682e - 17]
```

通过观察上面的分析结果，可以看出：t 值＝12.43054293，p 值＝9.13672682e-17，远小于 0.05，因此需要拒绝原假设。也就是说，该政策能引起股票价格的明显变化。

8.5　单样本方差假设检验的 Python 应用

方差用来反映波动情况，经常应用在金融市场波动等情形。单一总体方差的假设检验的基本步骤是首先提出原假设和备择假设，规定好检验的显著性水平，然后确定适当的检验统计量，并计算检验统计量的值，最后依据计算值和临界值的比较作出统计决策。

例 8-10　为了研究某基金的收益率波动情况，某课题组对该只基金的连续 50 天的收益率情况进行了调查研究，调查得到的数据经整理后如表 8-9 所示。试应用 Python 对该数据资料进行假设检验其方差（收益率波动）是否等于 1%（设定显著性水平为 5%）。

表 8-9　某基金的收益率波动情况

编号	收益率
1	0.564409196
2	0.264802098
3	0.947742641
4	0.276915401
5	0.118015848
…	…
48	−0.967873454
49	0.582328379
50	0.795299947

在目录 G:\2glkx\data 下建立 al8-4.xls 数据文件后，取数的命令如下：

```
import pandas as pd
import numpy as np
#读取数据并创建数据表,名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al8 - 4.xls '))
#查看数据表前 5 行的内容
data.head()
     bh     syl
   0  1  0.564409
   1  2  0.264802
   2  3  0.947743
   3  4  0.276915
   4  5  0.118016
#取收益率数据
import numpy as np
x = np.array(data[['syl']])
n = len(x)
#计算方差
s2 = np.var(x)
#计算卡方值
chisquare = (n - 1) * s2/0.01
print chisquare
1074.95071767
```

查表 $\chi^2_{0.025}$ = 56 (卡方关键值)

卡方统计值 1074.95071767>卡方关键值 56，卡方统计值落在拒绝区域，因此我们拒绝原假设，即该股票的方差显著地不等于 1%。

8.6　双样本方差假设检验的 Python 应用

双样本方差的假设检验是用来判断两个样本的波动情况是否相同，在金融市场领域应用研究中相当广泛。其基本步骤也是首先提出原假设和备择假设，规定好检验的显著性水平，然后确定适当的检验统计量，并计算检验统计量的值，最后依据计算值和临界值的比较

作出统计决策。

例 8-11 为了研究某两只基金的收益率波动情况是否相同，某课题组对该两只基金连续 20 天的收益率情况进行了调查研究，调查得到的数据经整理后如表 8-10 所示。试使用 Python 对该数据资料进行假设检验其方差是否相同(设定显著性水平为 5%)。

表 8-10 某两只基金的收益率波动情况

编号	基金 A 收益率	基金 B 收益率
1	0.424156	0.261075
2	0.898346	0.165021
3	0.521925	0.760604
4	0.841409	0.37138
5	0.211008	0.379541
…	…	
18	0.564409	0.967873
19	0.264802	0.582328
20	0.947743	0.7953

准备工作如下：

```
import pandas as pd
import numpy as np
from scipy import stats
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
```

在目录 G:\2glkx\data 下建立 al8-5.xls 数据文件后，取数的命令如下：

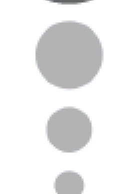
```
#读取数据并创建数据表,名称为 data。
df = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al8 - 5.xls'))
#查看数据表前 5 行的内容
df.head()
      returnA   returnB
   0  0.424156  0.261075
   1  0.898346  0.165021
   2  0.521925  0.760604
   3  0.841409  0.371380
   4  0.211008  0.379541
```

Python 中的 anova_lm()函数可完成两样本的 F 检验，即双样本方差的假设检验。

```
formula = 'returnA~returnB'       #~ 隔离因变量和自变量 (左边因变量,右边自变量 )
model = ols(formula,df).fit()     # 根据公式数据建模,拟合
results = anova_lm(model)         # 计算 F 和 P
print results
```

输入完后，按回车键，得到如下的分析结果。

```
             df     sum_sq  mean_sq          F  PR(>F)
returnB     1.0  0.000709  0.000709  0.007744  0.93085
Residual   18.0  1.648029  0.091557       NaN      NaN
```



通过观察上面的分析结果，可以看出：$F=0.007744$，p 值＝ 0.93085，远大于 0.05，因此需要接受原假设，也就是说，两只基金的收益率方差（波动）显著相同。

练　习　题

对本章例题数据文件，使用 Python 重新操作一遍，并理解命令结果的统计意义。



相关分析与一元回归数据分析的 Python 应用

在得到相关数据资料后，我们要对这些数据进行分析，研究各个变量之间的关系。相关分析是应用非常广泛的一种方法。它是不考虑变量之间的因果关系而只研究变量之间的相关关系的一种统计分析方法。本章首先介绍相关分析的基本理论及具体实例应用。

回归分析是经典的数据分析方法之一，应用范围非常广泛。它是研究分析某一变量受到其他变量影响的分析方法，其基本思想是以被影响变量为因变量，以影响变量为自变量，研究因变量与自变量之间的因果关系。本章主要介绍简单线性回归分析的基本理论及其具体实例应用。

9.1　相关分析基本理论

简单相关分析(bivariate)是最简单也是最常用的一种相关分析方法，其基本功能是可以研究变量之间的线性相关程度并用适当的统计指标表示出来。

9.1.1　简单相关系数的计算

两个随机变量(X,Y)的 n 个观测值为(x_i,y_i)，$i=1,2,\cdots,n$，则(X,Y)之间的相关系数计算公式如下：

$$r=\frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum(x_i-\bar{x})^2\sum(y_i-\bar{y})^2}} \tag{1}$$

其中 $\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i$，$\bar{y}=\frac{1}{n}\sum_{i=1}^{n}y_i$ 分别为随机变量 X 和 Y 的均值。

可以证明：$-1\leqslant r\leqslant 1$，即$|r|\leqslant 1$，于是有：

当$|r|=1$ 时，实际 y_i 完全落在回归直线上，y 与 x 完全线性相关；

当 $0<r<1$ 时，y 与 x 有一定的正线性相关，愈接近 1 则愈好；

当 $-1<r<0$ 时，y 与 x 有一定的负线性相关，愈接近 -1 则愈好。

9.1.2　简单相关系数的显著性检验

由于抽样误差的存在，当相关系数不为 0 时，不能说明两个随机变量 X 和 Y 之间的相关系数不为 0，因此需要对相关系数是否为 0 进行检验，即检验相关系数的显著性。

按照假设检验的步骤，简单相关系数显著性检验过程如下：

(1) 先建立原假设 H_0 和备择假设 H_1

H_0：$r=0$，相关系数为 0

H_1：$r\neq0$，相关系数不为 0

(2) 建立统计量 $t=r\sqrt{n-2}/\sqrt{1-r^2}$，其中 r 为相关系数，n 为样本容量。

(3) 给定显著水平，一般为 0.05。

(4) 计算统计量的值。

在 H_0 成立的条件下，$t=r\sqrt{n-2}/\sqrt{1-r^2}$，否定域 $\theta=\{|t|>t_{\alpha/2}(n-2)\}$。

(5) 统计决策

对于给定的显著性水平 α，查 t 分布表得临界值 $t_{\alpha/2}(n-2)$，将 t 值与临界值进行比较：

当 $|t|<t_{\alpha/2}(n-2)$，接受 H_0，表示总体的两变量之间线性相关性不显著；

当 $|t|\geqslant t_{\alpha/2}(n-2)$，拒绝 H_0，表示总体的两变量之间线性相关性显著（即样本相关系数的绝对值接近 1，并不是由于偶然机会所致）。

9.2　相关分析的 Python 应用

例 9-1　在研究广告费和销售额之间的关系时，我们搜集了某厂 1 月到 12 月各月广告费和销售额数据，如表 9-1 所示。试分析广告费和销售额之间的相关关系。

表 9-1　广告费和销售额数据　　单位：万元

月份	广告费	销售额
1	35	50
2	50	100
3	56	120
4	68	180
5	70	175
6	100	203
7	130	230
8	180	300
9	200	310
10	230	325
11	240	330
12	250	340

在目录 G:\2glkx\data 下建立 al9-1.xls 数据文件后，取数的命令如下：

```
import pandas as pd
import numpy as np
#读取数据并创建数据表,名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al9 - 1.xls '))
#查看数据表前 5 行的内容
data.head()
     time  adv   sale
   0    1  35    50
```

```
1    2  50  100
2    3  56  120
3    4  68  180
4    5  70  175
#取 adv 和 sale 数据
x = np.array(data[['adv']])
y = np.array(data[['sale']])
import scipy.stats.stats as stats
r = stats.pearsonr(x,y)[0]
print r
[ 0.96368169]
```

通过观察上面的结果，可以看到变量两两之间的相关系数，adv 和 sale 之间的相关系数为 0.96368169，也就是说，本例中变量之间相关性很高。

9.3 一元线性回归分析基本理论

9.3.1 一元线性回归分析模型

一元线性回归分析模型如下：

$$Y_i = b_0 + b_1 X_i + \varepsilon_i$$

其中：X 称为自变量，Y 称为因变量，ε 称为残差项或误差项。

给定若干的样本点(X_i, Y_i)，利用最小二乘法可以找到这样一条直线，它的截距为$\hat{b}_0$，斜率为$\hat{b}_1$，符号上面的帽子"^"表示"估计值"。因此我们得到回归结果如下：

$$\hat{Y}_i = \hat{b}_0 + \hat{b}_1 X_i$$

截距的含义是：$X=0$ 时 Y 的值。斜率的含义是：如果 X 增加 1 个单位，Y 增加几个单位。

回归的目的是为了预测因变量 Y，已知截距和斜率的估计值，如果得到了自变量 X 的预测值，我们就很容易求得因变量 Y 的预测值。

例 9-1 某公司的分析师根据历史数据，做了公司销售额增长率关于 GDP 增长率的线性回归分析，得到截距为－3.2%，斜率为 2，国家统计局预测今年 GDP 增长率为 9%，问该公司今年销售额增长率预计为多少？

解： $Y=-3.2\%+2X=-3.2\%+2*9\%=14.8\%$

9.3.2 一元线性回归的假设

任何模型都有假设前提，一元线性回归模型有以下 6 条假设。

1. 自变量 X 和因变量 Y 之间存在线性关系。

2. 残差项的期望值为 0。残差＝真实的 Y 值与预测的 Y 值之间的差，即预测的误差。期望值为 0 即有些点在回归线的上方，有些点在回归线的下方，且均匀围绕回归直线，这符合常理。

3. 自变量 X 与残差项不相关。残差项本身就是 Y 的变动中不能被 X 的变动所解释的

部分。

4. 残差项的方差为常数。这称为同方差性。如果残差项的方差不恒定，称为异方差性。

5. 残差项与残差项之间不相关。如果残差项与残差项之间相关，称为自相关或序列相关。

6. 残差项为正态分布的随机变量。

9.3.3　方差分析

做完了一个一元回归模型之后，我们通常想要知道回归模型做得好不好。方差分析可以用来评价回归模型的好坏。方差分析的结果是一张表，如表 9-2 所示。

表 9-2　方差分析

	自由度	平方和	均方和 MS
回归	$k=1$	回归平方和 RSS	回归均方和 MSR=RSS/k
误差	$n-2$	误差平方和 SSE	误差均方和 MSE=SSE/($n-2$)
总和	$n-1$	总平方和 SST	

我们可以从方差分析表里求得决定系数和估计的标准误，用来评价回归模型的好坏。

回归的自由度为 k，k 为自变量的个数。我们在一元线性回归，所以自变量的个数为 1。误差的自由度为 $n-2$，n 是样本量。总自由度为以上两个自由度之和。

总平方和 SST 代表总的变动，回归平方和 RSS 代表可以被回归方程解释（即可以被自变量解释）的变动，误差平方和 SSE 代表不被回归方程解释（即被残差所解释）的变动。总平方和为以上两个平方和之和。SST=RSS+SSE。

均方和等于各自的平方和除以各自的自由度。

几乎所有的统计软件都能输出方差分析表。有了方差分析表，就能很容易求得决定系数和估计的标准误。

9.3.4　决定系数

决定系数等于回归平方和除以总平方和，公式为

$$R^2=\frac{\text{RSS}}{\text{SST}}=1-\frac{\text{SSE}}{\text{SST}}$$

决定系数的含义是：X 的变动可以解释多少比例的 Y 的变动。例如 0.7 的含义是：X 的变动可以解释 70%的 Y 的变动。注意：是用 X 来解释 Y 的。

所以通俗地说，$R^2=\frac{\text{可以被解释的变动}}{\text{总的变动}}=1-\frac{\text{不可以被解释的变动}}{\text{总的变动}}$

显然，决定系数越大，表示回归模型越好。

另外，对于一元回归，决定系数还等于自变量和因变量的样本相关系数的平方，即 $R^2=r^2$。

9.3.5　估计的标准误

估计的标准误 SEE 等于残差均方和的平方根，公式为

$$\text{SEE} = \sqrt{\text{SSE}/(n-2)} = \sqrt{\text{MSE}}$$

SSE 是残差的平方和，MSE 就相当于残差的方差，而 SEE 就相当于残差的标准差。显然，估计的标准误越小，表示回归模型越好。

例 9-2 我们做了一个线性回归模型，得到如表 9-3 所示的方差分析表。

表 9-3 方 差 分 析

	自由度	平方和	均方和 MS
回归	1	8000	8000
误差	50	2000	40
总和	51	10000	

则决定系数和估计的标准误分别多少？

解：决定系数为 0.8，估计的标准误为 6.32。

9.3.6 回归系数的假设检验

回归系数的假设检验是指检验回归系数（截距和斜率）是否等于某个常数。通常要检验斜率系数是否等于 0（H_0：$b_1=0$），这称为斜率系数的显著性检验。如果不能拒绝原假设，即斜率系数没有显著的不等于 0，那就说明自变量 X 和因变量 Y 的线性相关性不大，回归是失败的。

这是一个 t 检验，t 统计量自由度为 $n-2$，计算公式为：$t=\dfrac{\hat{b}_1}{s_{\hat{b}_1}}$。

其中 $s_{\hat{b}_1}$ 为斜率系数的标准误。

例 9-3 我们做了一个线性回归模型，得到 $Y=0.2+1.4X$。截距系数的标准误为 0.4，斜率系数的标准误为 0.2，问：截距和斜率系数的显著性检验结果如何？设显著性水平为 5%。

解：截距系数的显著性检验：计算 t 统计量，$t=0.2/0.4=0.5<2$（t 检验的临界点），因此我们不能拒绝原假设，即认为截距系数没有显著的不等于 0。

其次是斜率系数的显著性检验：计算 t 统计量，$t=1.4/0.2=7>2$（t 检验的临界点），因此我们拒绝原假设，即认为斜率系数显著不等于 0。这说明我们的回归做得不错。

9.3.7 回归系数的置信区间

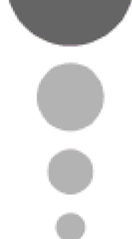
置信区间估计与假设检验本质上是一样的，一般公式为：点估计±关键值×点估计得标准差。回归系数的置信区间也是这样的。

斜率系数的置信区间为：$\hat{b}_1 \pm t_c s_{\hat{b}_1}$，其中，$t_c$ 是自由度为 $n-2$ 的 t 关键值。

例如我们做了一个线性回归模型，得到 $Y=0.2+1.4X$。截距系数的标准误为 0.4，斜率系数的标准误为 0.2，求截距和斜率系数的置信度为 95%的置信区间。

截距系数的置信区间，假设 n 充分大，5%的显著性水平的 t 关键值一般近似为 2，所以我们得到置信区间为：0.2±2×0.4，即[−0.6，1.0]。0 包括在置信区间中，所以我们认为截距系数没有显著地不等于 0。

斜率系数的置信区间：1.4±2×0.2，即[1.0，1.8]。0 没有包括在置信区间中，所以我



们认为斜率系数显著地不等于 0。

9.4　一元线性回归数据分析的 Python 应用

简单线性回归分析也称一元线性回归分析，是最简单也是最基本的一种回归分析方法。简单线性回归分析的特色是只涉及一个自变量，它主要用来处理一个因变量与一个自变量之间的线性关系，建立变量之间的线性模型并根据模型进行评价和预测。

例 9-4　某公司为研究销售人员数量对新产品销售额的影响，从其下属多家公司中随机抽取 10 个子公司，这 10 个子公司当年新产品销售额和销售人员数量统计数据如表 9-4 所示。试用简单回归分析方法研究销售人员数量对新产品销售额的影响。

表 9-4　新产品销售额和销售人员数量统计数据

地区	新产品销售额/万元	销售人员数量/人
1	385	17
2	251	10
3	701	44
4	479	30
5	433	22
6	411	15
7	355	11
8	217	5
9	581	31
10	653	36

9.4.1　应用 Python 的 statsmodels 工具作一元回归分析

在目录 G:\2glkx\data 下建立 al9-2. xls 数据文件后，取数的命令如下：

```
import pandas as pd
import numpy as np
#读取数据并创建数据表，名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al9 - 2.xls '))
data.head()
 dq   xse   rs
0   1   385   17
1   2   251   10
2   3   701   44
3   4   479   30
4   5   433   22
```

1. 对数据进行描述性分析

输入如下命令：

```
data.describe()
```

```
#此命令的含义是对销售额 xse、人数 rs 等变量进行描述性统计分析。
```

输入完后,按回车键,得到如下的分析结果:

```
              dq          xse         rs
count   10.00000    10.000000  10.000000
mean     5.50000   446.600000  22.100000
std      3.02765   160.224287  12.705642
min      1.00000   217.000000   5.000000
25%      3.25000   362.500000  12.000000
50%      5.50000   422.000000  19.500000
75%      7.75000   555.500000  30.750000
max     10.00000   701.000000  44.000000
```

通过观察上面的结果,可以得到很多信息,包括 2 个计数、2 个平均值、2 个标准差、2 个最小值、2 个第一百分位数、2 个第 2 百分位数、2 个第三百分位数、2 个最大值等。

更多信息描述如下。

(1) 最小值(Smallest)

变量 xse 最小值是 217.0。

变量 rs 最小值是 5.00。

(2) 百分位数

可以看出变量 xse 的第 1 个四分位数(25%)是 362.5,第 3 个四分位数(75%)是 555.5。变量 rs 的第 1 个四分位数(25%)是 12.00,第 3 个四分位数(75%)是 30.75。

(3) 平均值(Mean)

变量 xse 平均值的数据值分别是 446.6。

变量 rs 平均值的数据值分别是 22.10。

(4) 最大值(Largest)

变量 xse 最大值是 701.0。

变量 rs 最大值是 44.00。

2. 对数据进行相关分析

```
x = np.array(data[['rs']])
y = np.array(data[['xse']])
import scipy.stats.stats as stats
r = stats.pearsonr(x,y)[0]
#本命令的含义是对新产品销售额、销售人员人数等变量进行相关性分析
print r
```

输入完后,单击回车键,得到如下的分析结果:

```
[ 0.96990621]
```

通过观察上面的结果,可以看出销售额 xse 和人数 rs 之间的相关系数为 0.96990621,这说明两个变量之间存在很强的正相关关系,所以我们可以做回归分析。

除了上面介绍的通过数组进行相关分析外,还可通过如下的数据框来进行相关分析,代码如下:

```
data.corr()
```

得到如下结果：

```
          dq       xse        rs
dq   1.000000  0.218510  0.085207
xse  0.218510  1.000000  0.969906
rs   0.085207  0.969906  1.000000
```

可见，xse 与 rs 的相关系数是 0.969906，它们之间高度相关，因此可进一步做回归分析。

3. 一元回归分析的 Python 的 statsmodels 工具应用

一元回归分析的 Python 的 statsmodels 工具应用程序代码如下：

```
import statsmodels.api as sm
import pandas as pd
import numpy as np
#读取数据并创建数据表，名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al9-2.xls'))
data.head()
  dq  xse  rs
0  1  385  17
1  2  251  10
2  3  701  44
3  4  479  30
4  5  433  22
x = np.array(data[['rs']])
y = np.array(data[['xse']])
# model matrix with intercept
X = sm.add_constant(x)
#least squares fit
model = sm.OLS(y, X)
fit = model.fit()
print fit.summary()
```

得到如下结果：

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.941
Model:                            OLS   Adj. R-squared:                  0.933
Method:                 Least Squares   F-statistic:                     126.9
Date:                Fri, 23 Sep 2016   Prob (F-statistic):           3.46e-06
Time:                        20:16:31   Log-Likelihood:                -50.301
No. Observations:                  10   AIC:                             104.6
Df Residuals:                       8   BIC:                             105.2
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
```

```
------------------------------------------------------------------------------
const          176.2952       27.327       6.451      0.000       113.279      239.311
x1              12.2310        1.086      11.267      0.000         9.728       14.734
==============================================================================
Omnibus:                          0.718   Durbin-Watson:                   1.407
Prob(Omnibus):                    0.698   Jarque-Bera (JB):                0.588
Skew:                            -0.198   Prob(JB):                        0.745
Kurtosis:                         1.879   Cond. No.                         52.6
==============================================================================
```

通过观察上面的结果,可以看出模型的 F 值 $F=126.9$,p 值为 0,说明该模型整体上是非常显著的。模型的可决系数 R-squared=0.941,修正的可决系数 Adjusted R-squared=0.933,说明模型的解释能力是很强的。

模型的回归方程是:

$$\text{xse} = 12.2310 \times \text{rs} + 176.2952$$

变量 rs 的系数标准误是 1.086,t 值为 11.267,p 值为 0.000,系数是非常显著的。常数项的系数标准误是 27.327,t 值为 6.451,p 值为 0.000,系数是非常显著的。

```
#画线性回归图
pylab.scatter(x, y)
pylab.plot(x, fit.fittedvalues)
```

得到一元线性回归的图形,如图 9-1 所示。

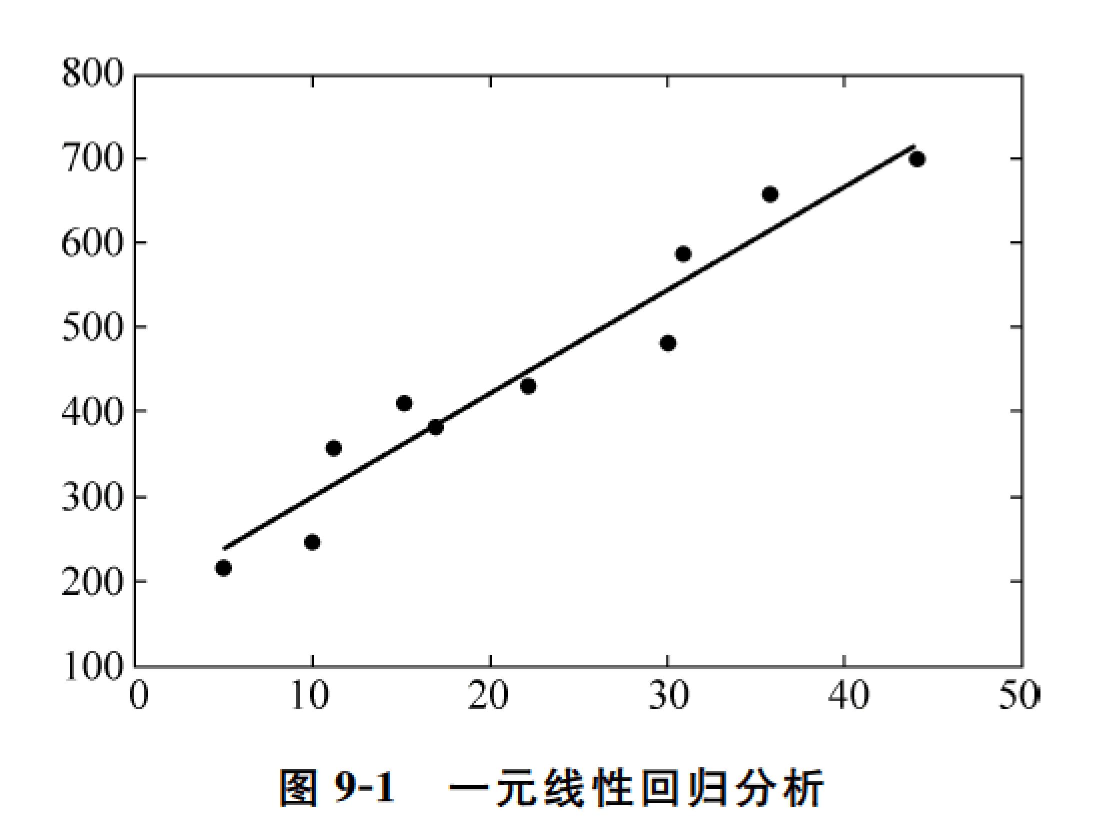


图 9-1　一元线性回归分析

9.4.2　应用 sklearn 工具作一元回归分析

下面应用 sklearn 工具做一元回归分析,输入如下命令:

```
from sklearn import linear_model
x = np.array(data[['rs']])
y = np.array(data[['xse']])
clf = linear_model.LinearRegression()
clf.fit (x,y)
clf.coef_
Out[5]: array([[ 12.2309863]])
clf.intercept_
Out[6]: array([ 176.2952027])
```

```
--------------------------------------------------------------------------------
const        176.2952      27.327      6.451      0.000      113.279      239.311
x1            12.2310       1.086     11.267      0.000        9.728       14.734
================================================================================
Omnibus:                     0.718   Durbin-Watson:                        1.407
Prob(Omnibus):               0.698   Jarque-Bera (JB):                     0.588
Skew:                       -0.198   Prob(JB):                             0.745
Kurtosis:                    1.879   Cond. No.                              52.6
================================================================================
```

通过观察上面的结果，可以看出模型的 F 值 $F=126.9$，p 值为 0，说明该模型整体上是非常显著的。模型的可决系数 R-squared＝0.941，修正的可决系数 Adjusted R-squared＝0.933，说明模型的解释能力是很强的。

模型的回归方程是：

$$\text{xse} = 12.2310 \times \text{rs} + 176.2952$$

变量 rs 的系数标准误是 1.086，t 值为 11.267，p 值为 0.000，系数是非常显著的。常数项的系数标准误是 27.327，t 值为 6.451，p 值为 0.000，系数是非常显著的。

```
#画线性回归图
pylab.scatter(x, y)
pylab.plot(x, fit.fittedvalues)
```

得到一元线性回归的图形，如图 9-1 所示。

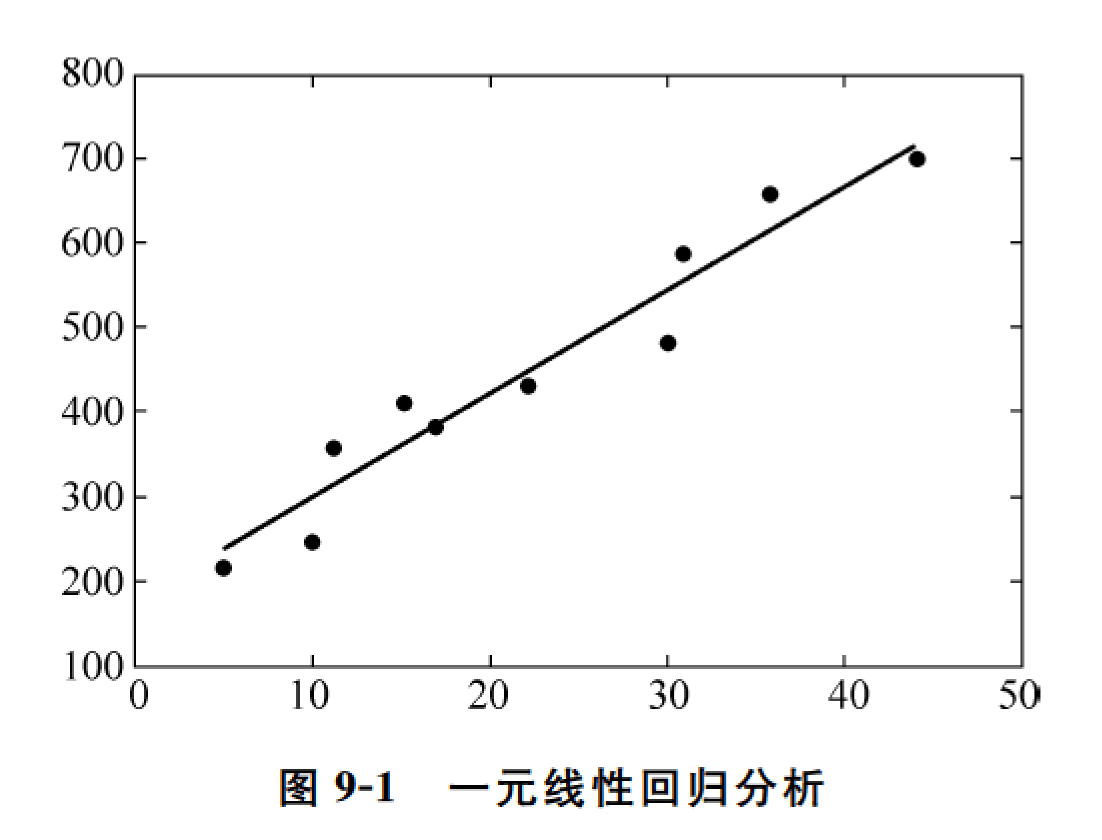
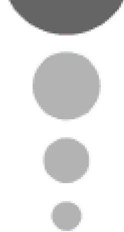


图 9-1 一元线性回归分析

9.4.2 应用 sklearn 工具作一元回归分析

下面应用 sklearn 工具做一元回归分析，输入如下命令：

```
from sklearn import linear_model
x = np.array(data[['rs']])
y = np.array(data[['xse']])
clf = linear_model.LinearRegression()
clf.fit (x,y)
clf.coef_
Out[5]: array([[ 12.2309863]])
clf.intercept_
Out[6]: array([ 176.2952027])
```




```
clf.score(x,y)
Out[7]: 0.9407180505879883
```

可见模型的可决系数 0.9407180505879883，说明模型的解释能力是很强的。

模型的回归方程是：

$$xse = 12.231rs + 176.295$$

若要求 rs=40 时相应 xse 预测值：

```
#输入自变量人数预测因变量
clf.predict(40)
Out[8]: array([[ 665.53465483]])
```

9.5　自相关性诊断的 Python 应用

例 9-5　某公司 1991—2005 年的开发经费和新产品利润数据如表 9-5 所示。利用回归分析开发经费对新产品利润的影响。

表 9-5　开发经费和新产品利润数据

开发费用/万元	新产品利润/万元
35	690
38	734
42	788
45	870
52	1038
65	1280
72	1434
81	1656
103	2033
113	2268
119	2451
133	2819
159	3431
198	4409
260	5885

在目录 G:\2glkx\data 下建立 al9-5. xls 数据文件后，使用如下命令读取数据：

```
import statsmodels.api as sm
import pandas as pd
import numpy as np
#读取数据并创建数据表,名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al9 - 5.xls '))
data.head()
   data.head()
      kf     lr
   0  35    690
```

```
   1  38    734
   2  42    788
   3  45    870
   4  52   1038
```

做 OLS 一元线性回归分析

```
x = np.array(data[['lr']])
y = np.array(data[['kf']])
# model matrix with intercept
X = sm.add_constant(x)
#least squares fit
model = sm.OLS(y, X)
fit = model.fit()
print fit.summary()
```

得到如下结果：

```
                           OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.998
Model:                            OLS   Adj. R-squared:                  0.997
Method:                 Least Squares   F-statistic:                     5535.
Date:                Sat, 29 Oct 2016   Prob (F-statistic):           1.74e-18
Time:                        15:05:28   Log-Likelihood:                -37.996
No. Observations:                  15   AIC:                             79.99
Df Residuals:                      13   BIC:                             81.41
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          9.2478      1.495      6.186      0.000       6.018      12.477
x1             0.0433      0.001     74.400      0.000       0.042       0.045
==============================================================================
Omnibus:                        1.182   Durbin-Watson:                   0.474
Prob(Omnibus):                  0.554   Jarque-Bera (JB):                1.011
Skew:                           0.515   Prob(JB):                        0.603
Kurtosis:                       2.255   Cond. No.                     4.54e+03
==============================================================================
```

从上可见，Durbin-Watson 统计量为 0.474，所以存在自相关。

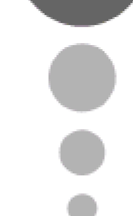
下面使用差分法来解决自相关问题。当模型存在自相关问题时，可以采用差分法来解决自相关问题。差分法的具体计算过程如下：

令 $\Delta y_i = y_i - y_{i-1}$，$\Delta x_{ij} = x_{ij} - x_{i-1j}$，$i=1,\cdots,n$，$j=1,\cdots,p$。利用 Δy_i 和 Δx_{ij} 数据，采取最小二乘法对下述回归模型的参数进行拟合，可以求出经验回归参数 β_j，$j=1,\cdots,p$。

$$\Delta y_i = \beta_0 + \beta_1 \Delta x_{i1} + \cdots + \beta_p \Delta x_{ip} + \varepsilon_i, \quad i = 1,\cdots,n$$

下面给出差分法消除自相关的 Python 代码：

```
x = np.array(data[['lr']])
y = np.array(data[['kf']])
# model matrix with intercept
```




```
X = sm.add_constant(x)
# least squares fit
model = sm.OLS(y, X)
fit = model.fit()
print fit.summary()
```

得到如下结果：

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.985
Model:                            OLS   Adj. R-squared:                  0.984
Method:                 Least Squares   F-statistic:                     777.9
Date:                Sat, 29 Oct 2016   Prob (F-statistic):           2.79e-12
Time:                        15:09:18   Log-Likelihood:                -29.448
No. Observations:                  14   AIC:                             62.90
Df Residuals:                      12   BIC:                             64.17
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          0.8469      0.791      1.071      0.305        -0.876     2.570
x1             0.0410      0.001     27.890      0.000         0.038     0.044
==============================================================================
Omnibus:                       12.469   Durbin-Watson:                   2.194
Prob(Omnibus):                  0.002   Jarque-Bera (JB):                8.230
Skew:                           1.508   Prob(JB):                       0.0163
Kurtosis:                       5.239   Cond. No.                         743.
==============================================================================
```

由上可见，Durbin-Watson 统计量为 2.194，自相关问题消除，说明采取差分法能够解决自相关问题。

练　习　题

对本章例题的数据文件，使用 Python 重新操作一遍。

第10章 多元回归数据分析的 Python 应用

10.1 多元线性回归分析基本理论

多元线性回归分析,也叫作多重线性回归分析,是最为常用的一种回归分析方法。多元线性回归分析涉及多个自变量,它用来处理一个因变量与多个自变量之间的线性关系,建立变量之间的线性模型并根据模型进行评价和预测。

10.1.1 多元线性回归模型

多元线性回归就是用多个自变量来解释因变量。多元线性回归模型如下:

$$Y_i = b_0 + b_1 X_{1i} + b_2 X_{2i} + \cdots + b_k X_{ki} + \varepsilon_i$$

利用最小二乘法可以找到这样一条直线:

$$\hat{Y}_i = \hat{b}_0 + \hat{b}_1 X_1 + \hat{b}_2 X_2 + \cdots + \hat{b}_k X_k$$

如果得到$\hat{b}_0$ 和多个$\hat{b}_j(j=1,\cdots,k)$以及所有自变量 $X_j(j=1,\cdots,k)$的预测值,就可求得因变量$\hat{Y}_i$ 的值。

例 10-1 某公司的分析师根据历史数据,做了公司销售额增长率关于 GDP 增长率和公司销售人员增长率的线性回归分析,得到截距为-3.2%,关于 GDP 增长率的斜率为 2,关于公司销售人员增长率的斜率为 1.2,国家统计局预测今年 GDP 增长率为 9%,公司销售部门预计公司销售人员今年将减少 20%。问:该公司今年销售额增长率预计为多少?

解:$Y=-3.2\%+2X_1+1.2X_2=-3.2\%+2\times 9\%+1.2\times(-20\%)=14.8\%$。

10.1.2 方差分析

与一元线性回归类似,多元线性回归的方差分析如表 10-1 所示。

表 10-1 方 差 分 析

	自由度	平方和	均方和 MS
回归	k	回归平方和 RSS	回归均方和 MSR=RSS/k
误差	$n-k-1$	误差平方和 SSE	误差均方和 MSE=SSE/$(n-k-1)$
总和	$n-1$	总平方和 SST	

我们可以从方差分析表 10-1 中求得决定系数和估计的标准误，用来评价回归模型的好坏。

回归的自由度为 k，k 为自变量的个数。误差的自由度为 $n-k-1$，n 是样本量。总自由度为以上两个自由度之和。

总平方和 SST 依然等于回归平方和与误差平方和之和。SST＝RSS＋SSE。

均方和等于各自的平方和除以各自的自由度，如表 10-1 所示。

有了上面的方差分析表，就能很容易求得决定系数和估计的标准误，以判断回归模型的好坏。

10.1.3　决定系数

决定系数等于回归平方和除以总平方和，公式为

$$R^2=\frac{\text{RSS}}{\text{SST}}=1-\frac{\text{SSE}}{\text{SST}}$$

和一元回归一样，多元回归的决定系数的含义仍然是：所有自变量 X 的变动可以解释多少比例的 Y 的变动。决定系数越大，表示回归模型越好。但是对于多元线性回归，随着自变量个数 k 的增加，决定系数总是变大，无论新增的自变量是否对因变量有解释作用。因此，我们就要调整决定系数如下：

$$\bar{R}^2=1-\frac{n-1}{n-k-1}(1-R^2)$$

调整后的决定系数不一定随着自变量个数 k 的增加而增大。因此调整后的决定系数能有效地比较不同自变量个数的回归模型的优劣。

关于调整后的决定系数，还要注意如下两点：

（1）调整后的决定系数总是小于等于未调整的决定系数。

（2）调整后的决定系数有可能小于 0。

10.1.4　估计的标准误

估计的标准误 SEE 等于残差均方和的平方根，公式为

$$\text{SEE}=\sqrt{\text{SSE}/(n-k-1)}=\sqrt{\text{MSE}}$$

显然，估计的标准误越小，表示回归模型越好。

10.1.5　回归系数的 *t* 检验和置信区间

与一元线性回归类似，回归系数的 t 检验是指检验回归系数是否等于某个常数。通常要检验斜率系数是否等于 0（H_0：$b_j=0$），这称为斜率系数的显著性检验。如果不能拒绝原假设，即斜率系数没有显著地不等于 0，那就说明自变量 X_j 和因变量 Y 的线性相关性不大，回归是失败的。

这是一个 t 检验，t 统计量自由度为 $n-k-1$，计算公式为：$t=\frac{\hat{b}_j}{s_{\hat{b}_j}}$。其中 $s_{\hat{b}_j}$ 为斜率系数的标准误。

斜率系数的置信区间为：$\hat{b}_j \pm t_c s_{\hat{b}_j}$。其中，$t_c$ 是自由度为 $n-k-1$ 的 t 关键值。

例如，我们做了一个二元线性回归模型，得到的结果如表 10-2 所示。

表 10-2　变量系数表

变量	系数	统计量
b_0	0.5	1.28
b_1	1.2	2.4
b_2	−0.3	0.92

斜率系数 b_1 的置信度为 95%的置信区间为多少？

由于统计量 $=2.4=t=\frac{\hat{b}_1}{s_{\hat{b}_1}}=\frac{1.2}{s_{\hat{b}_1}}$，$s_{\hat{b}_1}=1.2/2.4=0.5$。

$[1.2-2\times0.5, 1.2+2\times0.5]=[0.2, 2.2]$。

由于 0 没有包含在置信区间中，所以斜率系数 b_1 显著地不等于 0。

10.1.6　回归系数的 F 检验

回归系数的 F 检验用来检验斜率系数是否全部都等于 0。其原假设是所有斜率系数都等于 0，备择假设是至少有一个斜率系数不等于 0。

$$H_0: b_1=b_2=\cdots=b_k=0,\quad H_a: \text{至少有一个 } b_j\neq 0$$

F 统计量的分子自由度和分母自由度分别为 k 和 $n-k-1$，统计量的计算公式如下：

$$F=\frac{\text{MSR}}{\text{MSE}}=\frac{\text{RSS}/k}{\text{SSE}/(n-k-1)}$$

注意：F 检验看上去是双尾检验，但请当作单尾检验来做，其拒绝区域只在分布的右边一边。

回归系数的 t 检验是对单个斜率系数做检验，而回归系数的 F 检验是对全部斜率系数的检验。如果我们没有拒绝原假设，说明所有的斜率系数都没有显著地不等于 0，即所有自变量和因变量 Y 的线性相关性都不大，回归模型做得不好。如果我们能够拒绝原假设，说明至少有一个斜率系数显著地不等于 0，即至少有一个自变量可以解释 Y，回归模型做得不错。

例如我们抽取了一个样本量为 43 的样本，做了一个三元线性回归，得到 RSS=4500，SSE=1500M，以显著性水平为 0.05 检验是否至少有一个斜率系数显著地不等于 0。假设检验的结果如何？

$$\text{MSR}=\text{RSS}/k=4500/3=1500$$
$$\text{MSE}=\text{SSE}/(n-k-1)=1500/(43-3-1)=38.4$$
$$F=\text{MSR}/\text{MSE}=1500/38.4=39$$

查 F 统计表得关键值为 2.84。

由于 $2.84<39$，F 统计量落在拒绝区域，因此我们要拒绝原假设。

最后结论：至少有一个斜率系数显著地不等于 0。

10.1.7　虚拟变量

某些回归分析中，需要定性的使用自变量，称为虚拟变量。使用虚拟变量的目的是考察

不同类别之间是否存在显著差异。

虚拟变量的取值为 0 或 1，两类时，只需一个虚拟变量，如果 n 类，则需 $n-1$ 个虚拟变量。

例如，在研究工作水平同学历和工作年限的关系时，我们以 Y 表示工作水平，以 X_1 表示学历，以 X_2 表示工作年限，同时引进虚拟变量 D，其取值如下：

$$D=\begin{cases}1, & \text{男性}\\ 0, & \text{女性}\end{cases}$$

则可构造如下理论回归模型：

$$Y=\beta_0+\beta_1X_1+\beta_2X_2+\beta_3D+\varepsilon$$

为了模拟某商品销售量的时间序列的季节影响，我们需要引入 4－1＝3 个虚拟变量如下：

$$Q1=\begin{cases}1, & \text{如果为第 1 季度}\\ 0, & \text{其他情况}\end{cases};\quad Q2=\begin{cases}1, & \text{如果为第 2 季度}\\ 0, & \text{其他情况}\end{cases};$$

$$Q3=\begin{cases}1, & \text{如果为第 3 季度}\\ 0, & \text{其他情况}\end{cases}$$

则可构造如下理论回归模型：

$$Y=\beta_0+\beta_1Q_1+\beta_2Q_2+\beta_3Q_3+\varepsilon$$

10.2　多元线性回归数据分析的 Python 应用

例 10-2　为了检验美国电力行业是否存在规模经济，Nerlove(1963)搜集了 1955 年 145 家美国电力企业的总成本(TC)、产量(Q)、工资率(PL)、燃料价格(PF)及资本租赁价格(PK)的数据，如表 10-3 所示。试以总成本为因变量，以产量、工资率、燃料价格和资本租赁价格为自变量，利用多元线性回归分析方法研究它们之间的关系。

表 10-3　美国电力行业数据

编号	TC(百万美元)	Q(千瓦时)	PL(美元/千瓦时)	PF(美元/千瓦时)	PK (美元/千瓦时)
1	.082	2	2.09	17.9	183
2	.661	3	2.05	35.1	174
3	.99	4	2.05	35.1	171
4	.315	4	1.83	32.2	166
5	.197	5	2.12	28.6	233
6	.098	9	2.12	28.6	195
…	…	…	…	…	…
143	73.05	11796	2.12	28.6	148
144	139.422	14359	2.31	33.5	212
145	119.939	16719	2.3	23.6	162

在目录 G:\2glkx\data 下建立 al10-1.xls 数据文件后，取数的命令如下：

```
import pandas as pd
```

```
import numpy as np
#读取数据并创建数据表,名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al10 - 1.xls '))
data.head()
#前 5 条记录数据
      TC   Q   PL    PF    PK
0  0.082  2  2.09  17.9  183
1  0.661  3  2.05  35.1  174
2  0.990  4  2.05  35.1  171
3  0.315  4  1.83  32.2  166
4  0.197  5  2.12  28.6  233
```

1. 对数据进行描述性分析

输入如下命令：

```
data.describe()
#此命令的含义是对销售额 xse、人数 rs 等变量进行描述性统计分析。
```

输入完后，按回车键，得到如下的分析结果：

	TC	Q	PL	PF	PK
count	145.000000	145.000000	145.000000	145.000000	145.000000
mean	12.976097	2133.082759	1.972069	26.176552	174.496552
std	19.794577	2931.942131	0.236807	7.876071	18.209477
min	0.082000	2.000000	1.450000	10.300000	138.000000
25%	2.382000	279.000000	1.760000	21.300000	162.000000
50%	6.754000	1109.000000	2.040000	26.900000	170.000000
75%	14.132000	2507.000000	2.190000	32.200000	183.000000
max	139.422000	16719.000000	2.320000	42.800000	233.000000

通过观察上面的结果，可以得到很多信息，包括 5 个计数、5 个平均值、5 个标准差、5 个最小值、5 个第一百分位数、5 个第二百分位数、5 个第三百分位数、5 个最大值等。

通过观察上面的结果，可以得到很多信息，如 5 个最小值、第一百分位数、中位数、平均值、最大值等。更多的信息描述如下。

(1) 5 个最小值(Smallest)

变量总成本(TC)最小值是 0.082。

变量产量(Q)最小值是 2。

变量工资率(PL)最小值是 1.450。

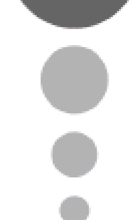
燃料价格(PF)最小值是 10.30。

资本租赁价格(PK)最小值是 138.0。

(2) 5 个百分位数

五个变量的第一百分位数分别是：2.382，279，1.760，21.30 ，162.0。

五个变量的第三百分位数分别是 14.132，2507，2.190，32.20，233.0。



(3) 5 个平均值(Mean)

五个变量的平均值分别是：12.976,2133,1.972,26.18,174.5。

(4) 5 个最大值(Largest)

5 个变量的最大值分别是：139.422,16719,2.320,42.80,233.0。

2. 用数组对数据做相关分析

```
y= np.array(data[['TC']])
x1 = np.array(data[['Q']])
x2 = np.array(data[['PL']])
x3 = np.array(data[['PF']])
x4 = np.array(data[['PK']])
import scipy.stats.stats as stats
r1 = stats.pearsonr(x1,y)[0]
r2 = stats.pearsonr(x2,y)[0]
r3 = stats.pearsonr(x3,y)[0]
r4 = stats.pearsonr(x4,y)[0]
print r1;print r2;print r3;print r4
```

输入完后，按回车键，得到如下的分析结果：

```
[ 0.9525037]
[ 0.25133754]
[ 0.03393519]
[ 0.027202]
```

通过观察上面的结果，可以看出总成本(TC)和产量(Q)、工资率(PL)、燃料价格(PF)、资本租赁价格(PK)之间的相关系数分别为 0.9525037、0.25133754、0.03393519、0.027202，这说明总成本 TC 变量与其他变量之间存在相关关系，所以我们可以回归分析。

3. 用数据框对数据做相关分析

用数组对数据做相关分析显得不太方便，下面用数据框做相关分析，就方便多了。代码如下：

```
data.corr()
```

得到如下结果：

	TC	Q	PL	PF	PK
TC	1.000000	0.952504	0.251338	0.033935	0.027202
Q	0.952504	1.000000	0.171450	-0.077349	0.002869
PL	0.251338	0.171450	1.000000	0.313703	-0.178145
PF	0.033935	-0.077349	0.313703	1.000000	0.125428
PK	0.027202	0.002869	-0.178145	0.125428	1.000000

从上可见，TC 与 Q 高度相关，而与其他的变量 PL,PF,PK 相关性要弱一些，尤其是与 PF 的相关性更弱。

3. 多元回归分析的 Python 的 statsmodels 工具应用

多元回归分析的 Python 的 statsmodels 工具应用程序代码如下：

```
import statsmodels.api as sm
import pandas as pd
import numpy as np
#读取数据并创建数据表,名称为 data。
data = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\al10-1.xls '))
data.head()
      TC     Q    PL    PF    PK
0   0.082    2   2.09  17.9  183
1   0.661    3   2.05  35.1  174
2   0.990    4   2.05  35.1  171
3   0.315    4   1.83  32.2  166
4   0.197    5   2.12  28.6  233
vars = ['TC','Q','PL','PF','PK']
df = data[vars]
#显示最后 5 条记录数据
print df.tail()
           TC       Q     PL    PF    PK
140    44.894    9956   1.68  28.8  203
141    67.120   11477   2.24  26.5  151
142    73.050   11796   2.12  28.6  148
143   139.422   14359   2.31  33.5  212
144   119.939   16719   2.30  23.6  162
```

下面生成设计矩阵。由于要建立的模型是 $y = BX$，因此需要分别求得 y 和 X 矩阵，而 dmatrices 就是做这个的，命令如下：

```
from patsy import dmatrices
y,X = dmatrices('TC~Q + PL + PF + PK',data = df,return_type = 'dataframe')
print y.head()
print X.head()
```

得到如下数据：

```
     TC
0   0.082
1   0.661
2   0.990
3   0.315
4   0.197
    Intercept    Q    PL     PF     PK
0         1.0  2.0  2.09   17.9  183.0
1         1.0  3.0  2.05   35.1  174.0
2         1.0  4.0  2.05   35.1  171.0
3         1.0  4.0  1.83   32.2  166.0
4         1.0  5.0  2.12   28.6  233.0
```

下面用 OLS 作普通最小二乘，fit 方法对回归方程进行估计，summary 保存计算的

结果。

```
import statsmodels.api as sm
model = sm.OLS(y, X)
fit = model.fit()
print fit.summary()
```

得到如下结果：

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     TC   R-squared:                       0.923
Model:                            OLS   Adj. R-squared:                  0.921
Method:                 Least Squares   F-statistic:                     418.1
Date:                Sat, 24 Sep 2016   Prob (F-statistic):           9.26e-77
Time:                        16:28:28   Log-Likelihood:                -452.47
No. Observations:                 145   AIC:                             914.9
Df Residuals:                     140   BIC:                             929.8
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept    -22.2210      6.587     -3.373      0.001      -35.245     9.197
Q              0.0064      0.000     39.258      0.000        0.006     0.007
PL             5.6552      2.176      2.598      0.010        1.352     9.958
PF             0.2078      0.064      3.242      0.001        0.081     0.335
PK             0.0284      0.027      1.073      0.285       -0.024     0.081
==============================================================================
Omnibus:                      135.057   Durbin-Watson:                   1.560
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4737.912
Skew:                           2.907   Prob(JB):                         0.00
Kurtosis:                      30.394   Cond. No.                     5.29e+04
```

通过观察上面的分析结果，可以看出：模型的 F 值$=418.11$，p 值$=0.0000$，说明模型整体上是非常显著的。模型的可决系数 R-squared$=0.923$，修正的可决系数 Adj R-squared$=0.921$，说明模型的解释能力是可以的。

模型的回归方程为

$$TC = 0.0064Q + 5.6552PL + 0.2078PF + 0.0284PK - 22.2210$$

变量 Q 的系数标准误是 0.000，t 值为 39.258，p 值为 0.000，系数是非常显著的。变量 PL 的系数标准误是 2.176，t 值为 2.598，p 值为 0.010，系数是非常显著的。变量 PF 的系数标准误是 0.064，t 值为 3.242，p 值为 0.001，系数是非常显著的。变量 PK 的系数标准误是0.027，t 值为 1.073，p 值为 0.285，系数是非常不显著的。常数项的系数标准误是6.587，t 值为-3.373，p 值为 0.001，系数是非常显著的。

综合上面的分析，可以看出：美国电力企业的总成本(TC)受到产量(Q)、工资率(PL)、燃料价格(PF)、资本租赁价格(PK)的影响，美国电力行业存在规模经济。

读者应注意上面的模型中，PK 的系数是不显著的。从前面的相关分析也可以看到，TC

与 PK 的相关系数很弱，只有 0.027202。下面把该变量剔除后重新进行回归分析，命令如下。

```
from patsy import dmatrices
y,X = dmatrices('TC~Q + PL + PF',data = df,return_type = 'dataframe')
import statsmodels.api as sm
model  =  sm.OLS(y, X)
fit  =  model.fit()
print fit.summary()
```

输入完上述命令后，按回车键，则得到如下的分析结果：

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     TC   R-squared:                       0.922
Model:                            OLS   Adj. R-squared:                  0.920
Method:                 Least Squares   F-statistic:                     556.5
Date:                Sat, 24 Sep 2016   Prob (F-statistic):           6.39e-78
Time:                        16:39:22   Log-Likelihood:                -453.06
No. Observations:                 145   AIC:                             914.1
Df Residuals:                     141   BIC:                             926.0
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept    -16.5443      3.928     -4.212      0.000      -24.309     -8.780
Q              0.0064      0.000     39.384      0.000        0.006      0.007
PL             5.0978      2.115      2.411      0.017        0.917      9.278
PF             0.2217      0.063      3.528      0.001        0.097      0.346
==============================================================================
Omnibus:                      142.387   Durbin-Watson:                   1.590
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5466.347
Skew:                           3.134   Prob(JB):                         0.00
Kurtosis:                      32.419   Cond. No.                     3.42e+04
==============================================================================
```

从上面分析结果可见，模型整体依旧是非常显著的。模型的可决系数以及修正的可决系数变化不大，说明模型的解释能力几乎没有变化。其他变量（含常数项的系数）都非常显著，模型接近完美。可以把回归结果作为最终的回归模型方程，即

$$\mathrm{TC} = 0.0064Q + 5.0978\mathrm{PL} + 0.2217\mathrm{PF} - 16.5443$$

从上面的分析可以看出，美国电力企业的总成本受到产量、工资率、燃料价格的影响。总成本随着这些变量的升高而升高、降低而降低。

值得注意的是：产量的增加引起总成本的相对变化是很小的。所以，从经济意义上说，美国电力行业存在规模经济。

10.3　多元回归分析的 Scikit-learn 工具应用

1. 使用 Pandas 来读取数据

Pandas 是一个用于数据探索、数据处理、数据分析的 Python 库。

```
import pandas as pd
# read csv file directly from a URL and save the results
data = pd.read_csv('http://www - bcf.usc.edu/～gareth/ISL/Advertising.csv', index_col = 0)
# display the first 5 rows
data.head()
Out[35]:
      TV  Radio   Newspaper  Sales
1  230.1   37.8        69.2   22.1
2   44.5   39.3        45.1   10.4
3   17.2   45.9        69.3    9.3
4  151.5   41.3        58.5   18.5
5  180.8   10.8        58.4   12.9
```

上面显示的结果类似一个电子表格，这个结构称为 Pandas 的数据帧(data frame)。

Pandas 的两个主要数据结构：Series 和 DataFrame。

(1) Series 类似于一维数组，它由一组数据以及一组与之相关的数据标签(即索引)组成。

(2) DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型。DataFrame 既有行索引也有列索引，它可被看作由 Series 组成的字典。

```
# display the last 5 rows
data.tail()
Out[36]:
         TV  Radio   Newspaper   Sales
196    38.2    3.7        13.8     7.6
197    94.2    4.9         8.1     9.7
198   177.0    9.3         6.4    12.8
199   283.6   42.0        66.2    25.5
200   232.1    8.6         8.7    13.4
# check the shape of the DataFrame(rows, colums)
data.shape
Out[37]: (200, 4)
```

特征：

TV——对于一个给定市场中的单一产品，用于电视上的广告费用(以千为单位)；

Radio——在广播媒体上投资的广告费用；

Newspaper——用于报纸媒体的广告费用；

响应变量：Sales——对应产品的销量。

在这个实例中，我们通过不同的广告投入，预测产品销量。因为响应变量是一个连续的值，所以这个问题是一个回归问题。数据集一共有 200 个观测值，每一组观测对应一个市场的情况。

```
import seaborn as sns #seaborn 程序包需要先安装
#安装命令: pip install seaborn
sns.pairplot(data, x_vars = ['TV','Radio','Newspaper'], y_vars = 'Sales', size = 7, aspect = 0.8)
```

得到如图 10-1 所示的图形。

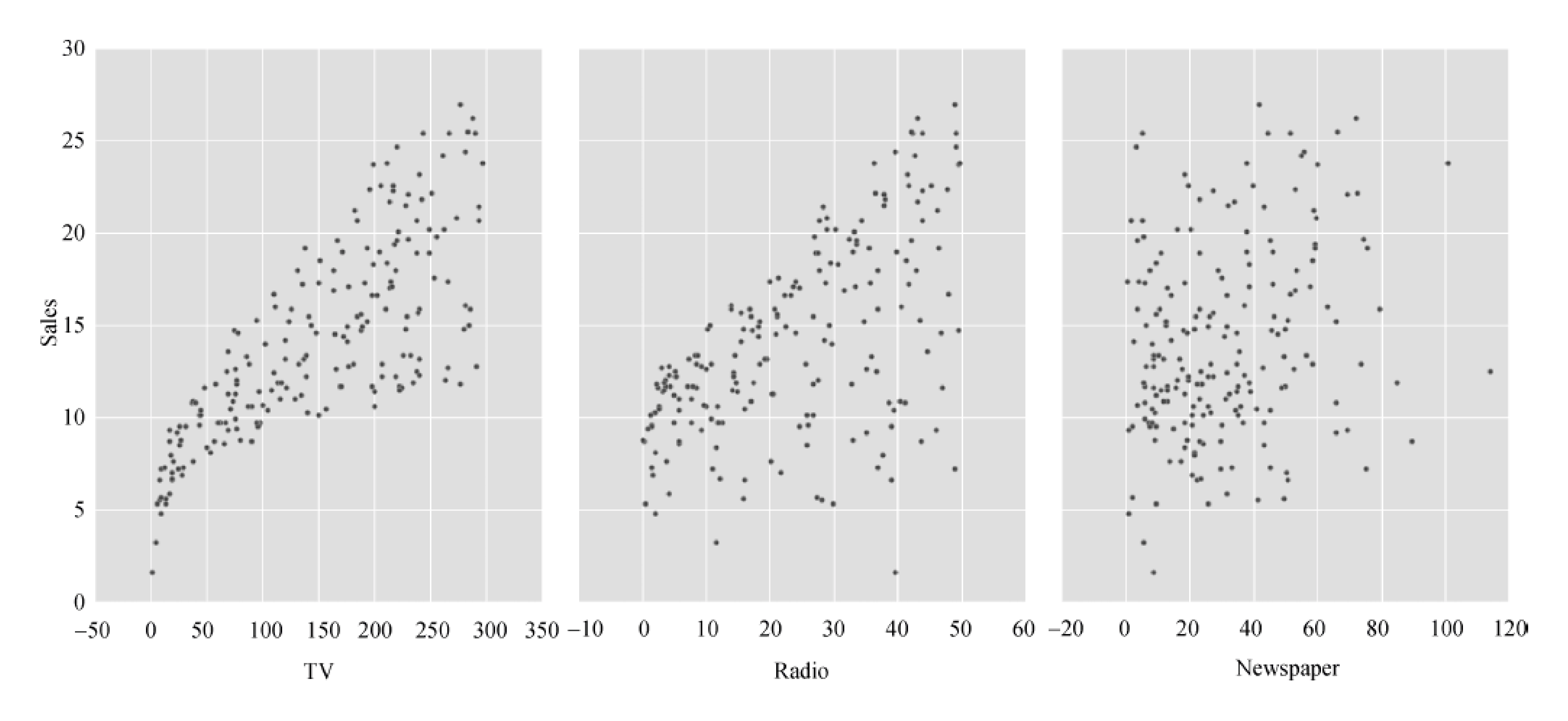


图 10-1 散点图

seaborn 的 pairplot 函数绘制 X 的每一维度和对应 Y 的散点图。通过设置 size 和 aspect 参数来调节显示的大小和比例。可以从图中看出,TV 特征和销量是有比较强的线性关系的,而 Radio 和 Sales 线性关系弱一些,Newspaper 和 Sales 线性关系更弱。通过加入一个参数 kind='reg',seaborn 可以添加一条最佳拟合直线和 95%的置信区间。

```
sns.pairplot(data, x_vars = ['TV','Radio','Newspaper'], y_vars = 'Sales', size = 7, aspect = 0.8,
kind = 'reg')
```

可以得到如图 10-2 所示的图形。

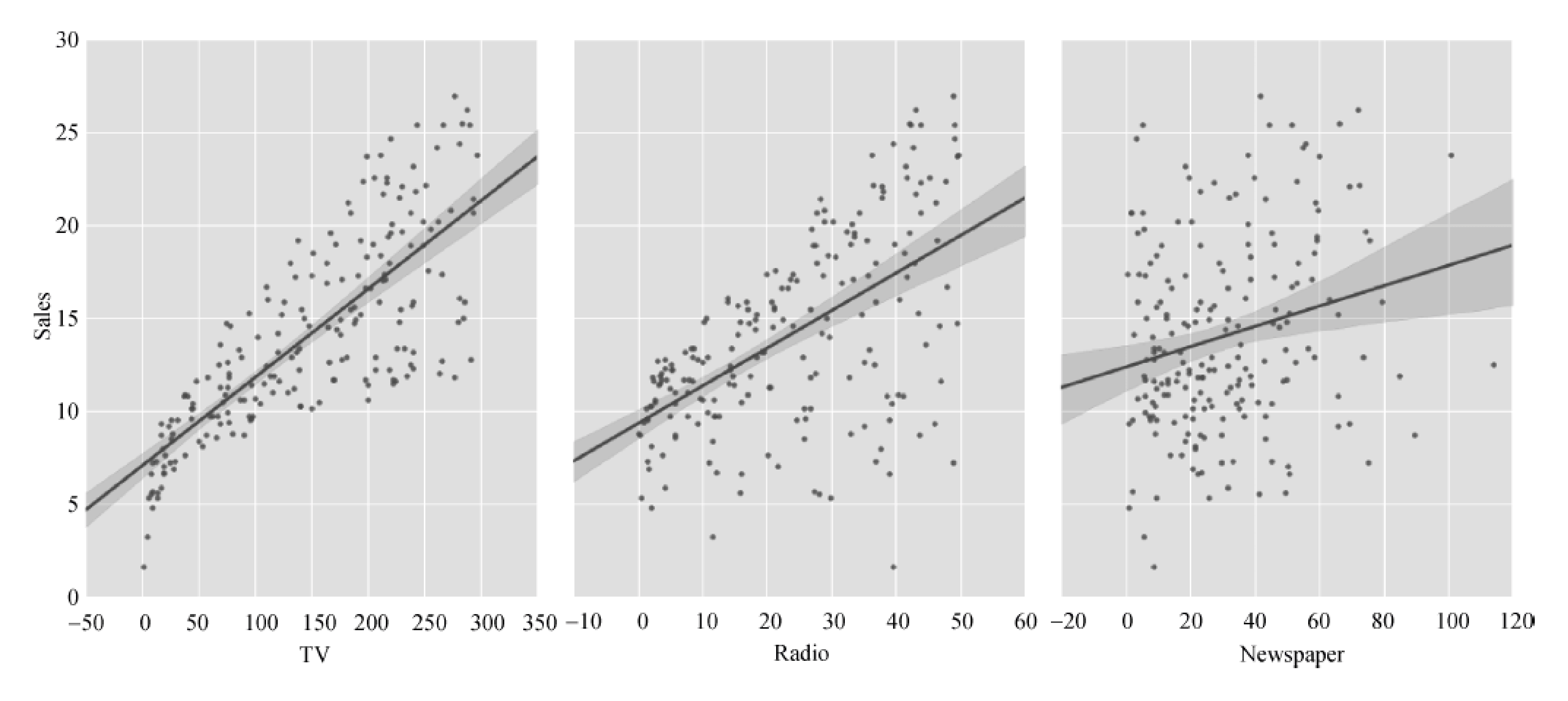


图 10-2 带有回归线的散点图

```
import seaborn as sns #seaborn 程序包需要先安装
#安装命令: pip install seaborn
sns.pairplot(data, x_vars = ['TV','Radio','Newspaper'], y_vars = 'Sales', size = 7, aspect = 0.8)
```

得到如图 10-1 所示的图形。

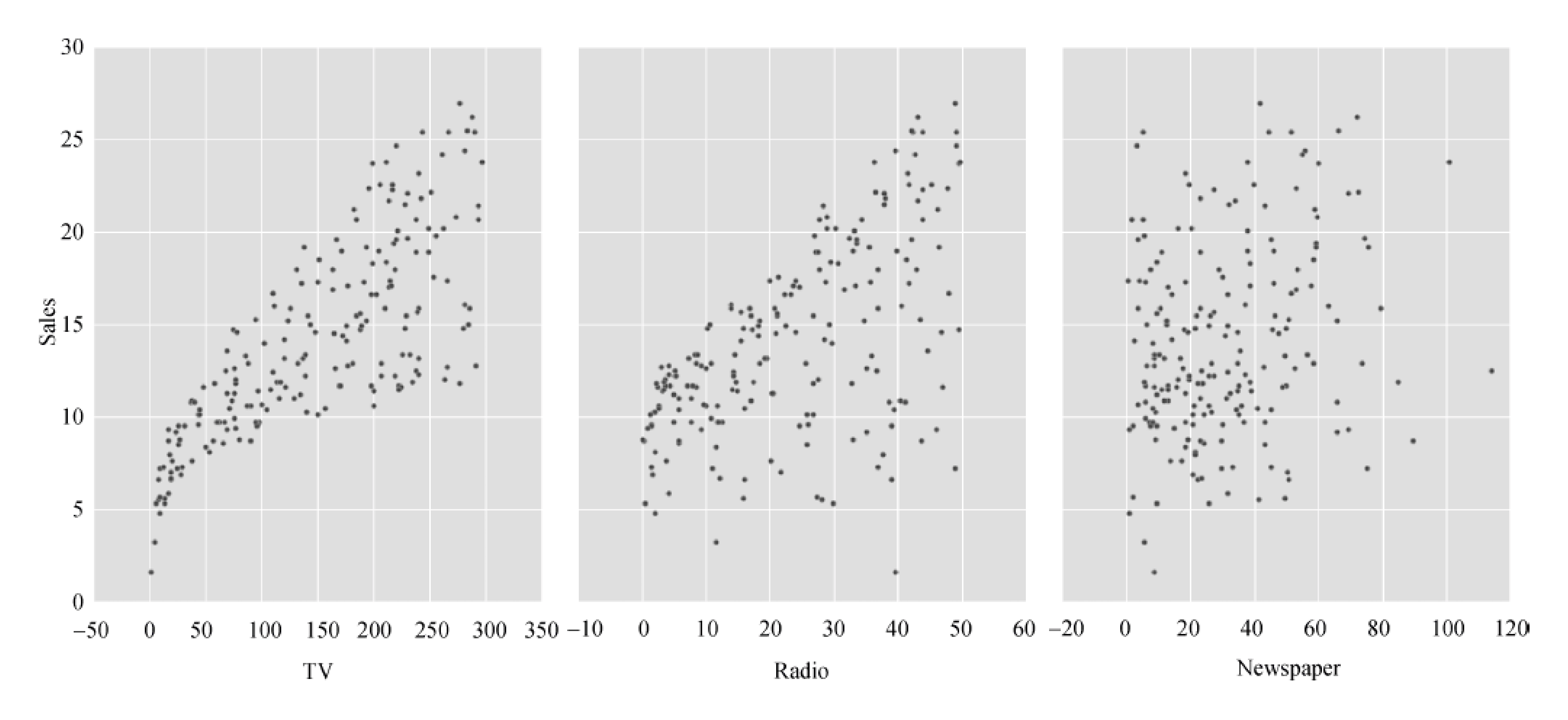


图 10-1　散点图

seaborn 的 pairplot 函数绘制 X 的每一维度和对应 Y 的散点图。通过设置 size 和 aspect 参数来调节显示的大小和比例。可以从图中看出,TV 特征和销量是有比较强的线性关系的,而 Radio 和 Sales 线性关系弱一些,Newspaper 和 Sales 线性关系更弱。通过加入一个参数 kind='reg',seaborn 可以添加一条最佳拟合直线和 95%的置信区间。

```
sns.pairplot(data, x_vars = ['TV','Radio','Newspaper'], y_vars = 'Sales', size = 7, aspect = 0.8,
kind = 'reg')
```

可以得到如图 10-2 所示的图形。

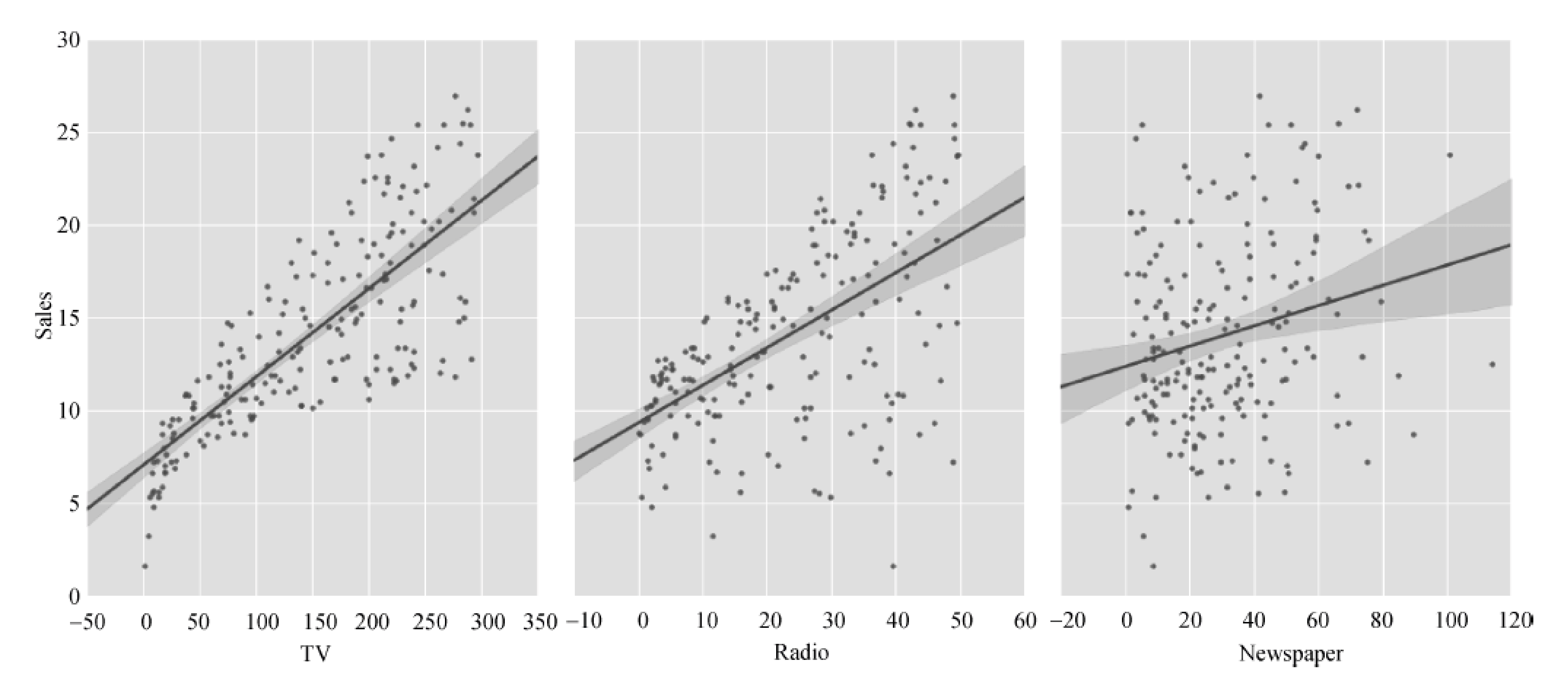


图 10-2　带有回归线的散点图

2. 线性回归模型

优点：快速、没有调节参数、易解释、可理解。

缺点：相比其他复杂一些的模型，其预测准确率不是太高，因为它假设特征和响应之间存在确定的线性关系，这种假设对于非线性的关系，线性回归模型显然不能很好地对这种数据建模。

线性模型表达式：$y=\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_n x_n$。

其中：y 是响应；β_0 是截距；β_1 是 x_1 的系数；依次类推。

在这个实例中：$y=\beta_0+\beta_1 \mathrm{TV}+\beta_2 \mathrm{Radio}+\cdots+\beta_n \mathrm{Newspaper}$。

(1) 使用 Pandas 来构建 ***X*** 和 ***y***

scikit-learn 要求 ***X*** 是一个特征矩阵，***y*** 是一个 NumPy 向量。

Pandas 构建在 NumPy 之上，因此，***X*** 可以是 Pandas 的 DataFrame，y 可以是 Pandas 的 Series，scikit-learn 可以理解这种结构。

```
# create a python list of feature names
feature_cols = ['TV', 'Radio', 'Newspaper']
# use the list to select a subset of the original DataFrame
X = data[feature_cols]
# equivalent command to do this in one line
X = data[['TV', 'Radio', 'Newspaper']]

# print the first 5 rows
X.head()
Out[41]:
      TV  Radio  Newspaper
1  230.1  37.8        69.2
2   44.5  39.3        45.1
3   17.2  45.9        69.3
4  151.5  41.3        58.5
5  180.8  10.8        58.4
# check the type and shape of X
print type(X)
print X.shape
<class 'pandas.core.frame.DataFrame'>
(200, 3)
# select a Series from the DataFrame
y = data['Sales']
# equivalent command that works if there are no spaces in the column name
y = data.Sales
# print the first 5 values
y.head()
Out[45]:
1  22.1
2  10.4
3   9.3
4  18.5
```

```
5   12.9
Name: Sales, dtype: float64
print type(y)
print y.shape
<class 'pandas.core.series.Series'>
(200,)
```

(2) 构造训练集和测试集

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
# default split is 75% for training and 25% for testing
print X_train.shape
print y_train.shape
print X_test.shape
print y_test.shape
(150, 3)
(150,)
(50, 3)
(50,)
```

(3) 线性回归分析的 Scikit-learn 工具应用

```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X_train, y_train)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
print linreg.intercept_
print linreg.coef_
2.87696662232
[ 0.04656457 0.17915812 0.00345046]
# pair the feature names with the coefficients
zip(feature_cols, linreg.coef_)
Out[49]:
[('TV', 0.046564567874150288),
 ('Radio', 0.17915812245088841),
 ('Newspaper', 0.0034504647111803788)]
```

因此回归直线方程为:

$$y = 2.88 + 0.0466\text{TV} + 0.179\text{Radio} + 0.00345\text{Newspaper}$$

如何解释各个特征对应的系数的意义?

对于给定了 Radio 和 Newspaper 的广告投入,如果在 TV 广告上每多投入 1 个单位,对应销量将增加 0.0466 个单位。更明确一点说,假如其他两个媒体投入固定,在 TV 广告上每增加 1000 美元(因为单位是 1000 美元),公司的销售量将增加 46.6 美元(因为单位是 1000)。

(4) 预测

```
y_pred = linreg.predict(X_test)
print y_pred
[ 21.70910292  16.41055243   7.60955058  17.80769552  18.6146359
  23.83573998  16.32488681  13.43225536   9.17173403  17.333853
```

```
  14.44479482   9.83511973  17.18797614  16.73086831  15.05529391
  15.61434433  12.42541574  17.17716376  11.08827566  18.00537501
   9.28438889  12.98458458   8.79950614  10.42382499  11.3846456
  14.98082512   9.78853268  19.39643187  18.18099936  17.12807566
  21.54670213  14.69809481  16.24641438  12.32114579  19.92422501
  15.32498602  13.88726522  10.03162255  20.93105915   7.44936831
   3.64695761   7.22020178   5.9962782   18.43381853   8.39408045
  14.08371047  15.02195699  20.35836418  20.57036347  19.60636679]
```

3. 回归问题的评价测度

下面介绍三种常用的针对回归问题的评价测度。

```
# define true and predicted response values
true = [100, 50, 30, 20]
pred = [90, 50, 50, 30]
```

（1）平均绝对误差(mean absolute error，MAE)

$$MAE = \frac{\sum |y_i - \hat{y}_i|}{n}$$

（2）均方误差(mean squared error，MSE)

$$MSE = \frac{\sum (y_i - \hat{y}_i)^2}{n}$$

（3）均方根误差(root mean squared error，RMSE)

$$RMSE = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n}}$$

```
from sklearn import metrics
import numpy as np
# calculate MAE by hand
print"MAE by hand:",(10 + 0 + 20 + 10)/4.
# calculate MAE using scikit-learn
print "MAE:",metrics.mean_absolute_error(true, pred)
# calculate MSE by hand
print "MSE by hand:",(10 ** 2 + 0 ** 2 + 20 ** 2 + 10 ** 2)/4.
# calculate MSE using scikit-learn
print "MSE:",metrics.mean_squared_error(true, pred)
# calculate RMSE by hand
print "RMSE by hand:",np.sqrt((10 ** 2 + 0 ** 2 + 20 ** 2 + 10 ** 2)/4.)
# calculate RMSE using scikit-learn
print "RMSE:",np.sqrt(metrics.mean_squared_error(true, pred))
```

得到如下结果：

```
MAE by hand: 10.0
MAE: 10.0
MSE by hand: 150.0
MSE: 150.0
RMSE by hand: 12.2474487139
RMSE: 12.2474487139
```

```
计算 Sales 预测的 RMSE
print np.sqrt(metrics.mean_squared_error(y_test, y_pred))
1.40465142303
```

4. 特征选择

在前面图 10-1 所示的图形展示中，我们看到 Newspaper 和销量之间的线性关系比较弱。现在我们移除这个特征，看看线性回归预测的结果的 RMSE 如何？

```
feature_cols = ['TV', 'Radio']
X = data[feature_cols]
y = data.Sales
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
linreg.fit(X_train, y_train)
y_pred = linreg.predict(X_test)
print np.sqrt(metrics.mean_squared_error(y_test, y_pred))
1.38790346994
```

将 Newspaper 这个特征移除之后，得到 RMSE 变小了，说明 Newspaper 特征不适合作为预测销量的特征，这样可以得到新的模型。还可以通过不同的特征组合得到新的模型，看看最终的误差如何。

10.4 稳健线性回归分析 Python 应用

以职业声望数据集为例，其中包括 income（收入），education（教育），prestige（声望）。准备工作如下：

```
from __future__ import print_function
from statsmodels.compat import lmap
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.graphics.api import abline_plot
from statsmodels.formula.api import ols, rlm
##取数据集
prestige = sm.datasets.get_rdataset("Duncan", "car", cache=True).data
##显示前 5 条
print(prestige.head(5))
            type  income  education  prestige
accountant  prof      62         86        82
pilot       prof      72         76        83
architect   prof      75         92        90
author      prof      55         90        76
chemist     prof      64         86        90
##稳健线性回归分析
rlm_model = rlm('prestige ~ income + education', prestige).fit()
print(rlm_model.summary())
```

得到结果如下：

```
                    Robust Linear Model Regression Results
==============================================================================
Dep. Variable:               Prestige   NO. Observations:                   45
Model:                            RLM   Df Residuals:                       42
Method:                          IRLS   Df Model:                            2
Norm:                          HuberT
Scale Est:                        mad
Cov Type:                          H1
Date:                Fri, 28 Oct 2016
Time:                        16:24:51
No. Iterations:                    18
==============================================================================
                 coef     std err        z      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept     -7.1107      3.879    -1.833      0.067     -14.713      -0.492
income         0.7015      0.109     6.456      0.000       0.489       0.914
education      0.4854      0.089     5.441      0.000       0.311       0.660
==============================================================================
Omnibus:                      142.387   Durbin-Watson:                   1.590
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5466.347
Skew:                           3.134   Prob(JB):                         0.00
Kurtosis:                      32.419   Cond. No.                     3.42e+04
==============================================================================
```

10.5　逻辑 Logistic 回归分析 Python 应用

10.5.1　相关理论

线性回归模型是定量分析中最常用的统计分析方法，但线性回归分析要求响应变量是连续型变量。在实际研究中，尤其适合在社会、经济数据的统计分析中，研究非连续型的响应变量，即分类响应变量。

在研究两元分类响应变量与诸多自变量间的相互关系时，常选用 Logistic 回归模型。

将两元分类响应变量 Y 的一个结果记为“成功”，另一个结果记为“失败”，分别用 0 或 1 表示。对响应变量 Y 有影响的 p 个自变量(解释变量)记为 $X_1,\cdots,X_p$。在 m 个自变量的作用下出现“成功”的条件概率记为 $p(Y=1|X_1,\cdots,X_p)$那么 Logistic 回归模型表示为

$$p=\frac{\exp(\beta_0+\beta_1x_1+\cdots+\beta_px_p)}{1+\exp(\beta_0+\beta_1x_1+\cdots+\beta_px_p)} \tag{1}$$

β_0 称为常数项，$\beta_1,\cdots,\beta_p$ 称为 Logistic 回归模型的回归系数。

从上式(1)可以看出，Logistic 回归模型是一个非线性的回归模型，自变量 $x_j, j=1,\cdots,p$ 可以是连续变量，也可以是分类变量或哑变量。对自变量 x_j 任意取值 $\beta_0+\beta_1x_1+\cdots+\beta_px_p$ 总落在$(-\infty,\infty)$中，上式(1)的比值即 p 值，总在 0 到 1 之间变化，这就是 Logistic 回归模型的合理性所在。

对公式(1)做 logit 变换，Logistic 回归模型可以写成下列线性形式：

$$\text{logit}(p) = \text{In}\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

这样就可以使用线性回归模型对参数 $\beta_j, j=1,\cdots,p$ 进行估计。

10.5.2 Python 应用

程序包的准备：

```
import numpy as np
import statsmodels.api as sm
```

程序包内含数据导入：

```
spector_data = sm.datasets.spector.load()
spector_data.exog = sm.add_constant(spector_data.exog, prepend = False)
```

数据展示：

```
print(spector_data.exog[:5,:])
[[ 2.66   20.    0.     1. ]
 [ 2.89   22.    0.     1. ]
 [ 3.28   24.    0.     1. ]
 [ 2.92   12.    0.     1. ]
 [ 4.     21.    0.     1. ]]
print(spector_data.endog[:5])
[ 0. 0. 0. 0. 1.]
```

逻辑回归分析代码如下：

```
logit_mod = sm.Logit(spector_data.endog, spector_data.exog)
logit_res = logit_mod.fit(disp = 0)
print('Parameters: ', logit_res.params)
```

得到结果如下：

```
('Parameters: ', array([ 2.82611259, 0.09515766, 2.37868766, -13.02134686]))
```

10.6 广义线性回归分析 Python 应用

10.6.1 相关理论

Logistic 回归模型属于广义线性模型的一种，它是通常的正态线性回归模型的推广，它要求响应变量只能通过线性形式依赖于解释变量。上述推广体现在两个方面：

(1) 通过一个连续函数 $\varphi(E(Y))=\beta_0+\beta_1 x_1+\cdots+\beta_p x_p$；

(2) 通过一个误差函数，说明广义线性模型的最后一部分随机项。

表 10-4 给出了广义线性模型中常见的连续函数和误差函数。可见，若连接函数为恒等变换，误差函数为正态分布，则得到通常的正态线性模型。

表 10-4　常见的连接函数和误差函数

变换	连接函数	回归模型	典型误差函数
恒等	$\varphi(x)=x$	$E(y)=X'\beta$	正态分布
logit	$\varphi(x)=\mathrm{logit}(x)$	$\ln git(E(y))=X'\beta$	二项分布
对数	$\varphi(x)=\ln(x)$	$\ln(E(y))=X'\beta$	泊松分布
逆(倒数)	$\varphi(x)=1/x$	$1/E(y)=X'\beta$	伽马分布

Python 的 statsmodels 程序包提供了各种拟合和计算广义线性模型的函数。

正态分布拟合和计算调用格式为：

```
import statsmodels.api as sm
gauss_log = sm.GLM(lny,X,family = sm.families.Gaussian())
res = gauss_log .fit()
print(res.summary())
```

二项分布拟合和计算调用格式为：

```
import statsmodels.api as sm
glm_binom = sm.GLM(data.endog, data.exog, family = sm.families.Binomial())
res = glm_binom.fit()
print(res.summary())
```

泊淞分布拟合和计算调用格式为：

```
import statsmodels.api as sm
glm_poisson = sm.GLM(data.endog, data.exog, family = sm.families.Poisson())
res = glm_poisson.fit()
print(res.summary())
```

伽马分布拟合和计算调用格式为：

```
import statsmodels.api as sm
glm_gamma = sm.GLM(data2.endog, data2.exog, family = sm.families.Gamma())
res = glm_gamma.fit()
print(res.summary())
```

下面以二项分布函数为例，来说明 Python 广义回归分析的应用。

10.6.2　Python 应用

程序包的准备：

```
import numpy as np
import statsmodels.api as sm
from scipy import stats
from matplotlib import pyplot as plt
```

程序包内含数据导入：

```
data = sm.datasets.star98.load()
data.exog = sm.add_constant(data.exog, prepend = False)
```

数据展示：

```
print(spector_data.exog[:5,:])
[[ 2.66 20.    0.    1.  ]
 [ 2.89 22.    0.    1.  ]
 [ 3.28 24.    0.    1.  ]
 [ 2.92 12.    0.    1.  ]
 [ 4.   21.    0.    1.  ]]
print(data.exog[:2,:])
[[ 3.43973000e+01  2.32993000e+01  1.42352800e+01  1.14111200e+01
   1.59183700e+01  1.47064600e+01  5.91573200e+01  4.44520700e+00
   2.17102500e+01  5.70327600e+01  0.00000000e+00  2.22222200e+01
   2.34102872e+02  9.41688110e+02  8.69994800e+02  9.65065600e+01
   2.53522420e+02  1.23819550e+03  1.38488985e+04  5.50403520e+03
   1.00000000e+00]
 [ 1.73650700e+01  2.93283800e+01  8.23489700e+00  9.31488400e+00
   1.36363600e+01  1.60832400e+01  5.95039700e+01  5.26759800e+00
   2.04427800e+01  6.46226400e+01  0.00000000e+00  0.00000000e+00
   2.19316851e+02  8.11417560e+02  9.57016600e+02  1.07684350e+02
   3.40406090e+02  1.32106640e+03  1.30502233e+04  6.95884680e+03
   1.00000000e+00]]
```

二项分布函数的广义回归分析代码如下：

```
import statsmodel s.api as sm
glm_binom = sm.GLM(data.endog, data.exog, family=sm.families.Binomial())
res = glm_binom.fit()
print(res.summary())
```

结果如下：

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:          ['y1', 'y2']   No. Observations:                  303
Model:                           GLM   Df Residuals:                      282
Model Family:               Binomial   Df Model:                           20
Link Function:                 logit   Scale:                             1.0
Method:                         IRLS   Log-Likelihood:                -2998.6
Date:               Tue, 21 Jun 2016   Deviance:                       4078.8
Time:                       13:14:17   Pearson chi2:                     9.60
No. Iterations:                    5
==============================================================================
                 coef    std err      z     P>|z|    [0.025     0.975]
------------------------------------------------------------------------------
x1        -0.0168      0.000     -38.749    0.000     -0.018     -0.016
x2         0.0099      0.001      16.505    0.000      0.009      0.011
x3        -0.0187      0.001     -25.182    0.000     -0.020     -0.017
x4        -0.0142      0.000     -32.818    0.000     -0.015     -0.013
x5         0.2545      0.030       8.498    0.000      0.196      0.313
x6         0.2407      0.057       4.212    0.000      0.129      0.353
x7         0.0804      0.014       5.775    0.000      0.053      0.108
```

```
x8       -1.9522       0.317      -6.162      0.000      -2.573      -1.331
x9       -0.3341       0.061      -5.453      0.000      -0.454      -0.214
x10      -0.1690       0.033      -5.169      0.000      -0.233      -0.105
x11       0.0049       0.001       3.921      0.000       0.002       0.007
x12      -0.0036       0.000     -15.878      0.000      -0.004      -0.003
x13      -0.0141       0.002      -7.391      0.000      -0.018      -0.010
x14      -0.0040       0.000      -8.450      0.000      -0.005      -0.003
x15      -0.0039       0.001      -4.059      0.000      -0.006      -0.002
x16       0.0917       0.015       6.321      0.000       0.063       0.120
x17       0.0490       0.007       6.574      0.000       0.034       0.064
x18       0.0080       0.001       5.362      0.000       0.005       0.011
x19       0.0002    2.99e-05       7.428      0.000       0.000       0.000
x20      -0.0022       0.000      -6.445      0.000      -0.003      -0.002
const     2.9589       1.547       1.913      0.056      -0.073       5.990
==============================================================================
```

练　习　题

对例题中的数据文件，使用 Python 的 OLS 工具和 scikit-learn 工具重新操作一遍。



机器学习数据分析的 Python 应用

11.1 机器学习算法分类

一般来说,机器学习算法有以下三类。

1. 监督式学习算法

这类算法由一个目标变量或结果变量(或因变量)组成。这些变量由已知的一系列预示变量(自变量)预测而来。利用这一系列变量,可以生成一个将输入值映射到期望输出值的函数。这个训练过程会一直持续,直到模型在训练数据上获得期望的精确度。监督式学习的例子有:线性回归、决策树、随机森林算法、K 最近邻算法、逻辑回归等。

2. 非监督式学习算法

这类算法没有任何目标变量或结果变量要预测或估计。它用在不同的组内聚类分析。这种分析方式被广泛地用来细分客户,根据干预的方式分为不同的用户组。非监督式学习的例子有:关联算法、K 均值算法等。

3. 强化学习算法

这类算法训练机器进行决策。原理是这样的:机器被放在一个能让它通过反复试错来训练自己的环境中。机器从过去的经验中进行学习,并且尝试利用了解最透彻的知识作出精确的商业判断。强化学习算法的例子有马尔可夫决策过程。

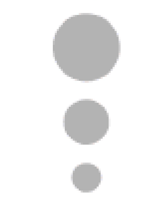
11.2 常见的机器学习算法及其 Python 代码

常见的机器学习算法有:(1)线性回归;(2)逻辑回归;(3)决策树;(4)支持向量机(SVM)分类;(5)朴素贝叶斯分类;(6)K 最近邻算法;(7)K 均值算法;(8)随机森林算法;(9)降维算法;(10)Gradient Boost 和 Adaboost 算法。

下面我们对上面的机器学习算法逐一介绍,并给出其主要的 Python 代码。

11.2.1 线性回归

线性回归通常用于根据连续变量估计实际数值(如房价、呼叫次数、总销售额等)。我们



通过拟合最佳直线来建立自变量和因变量的关系。这条最佳直线叫作回归线，并且用 $y=ax+b$ 这一线性等式来表示。

假设在不问对方体重的情况下，让一个五年级的孩子按体重从轻到重的顺序对班上的同学排序，你觉得这个孩子会怎么做？他很可能会目测人们的身高和体形，综合这些可见的参数来排列他们。这是现实生活中使用线性回归的例子。实际上，这个孩子发现了身高和体形、体重有一定的关系，这个关系看起来很像上面的等式。在这个等式中：

y：因变量；

x：自变量；

a：斜率；

b：截距。

系数 a 和 b 可以通过最小二乘法获得。

如图 11-1 所示。

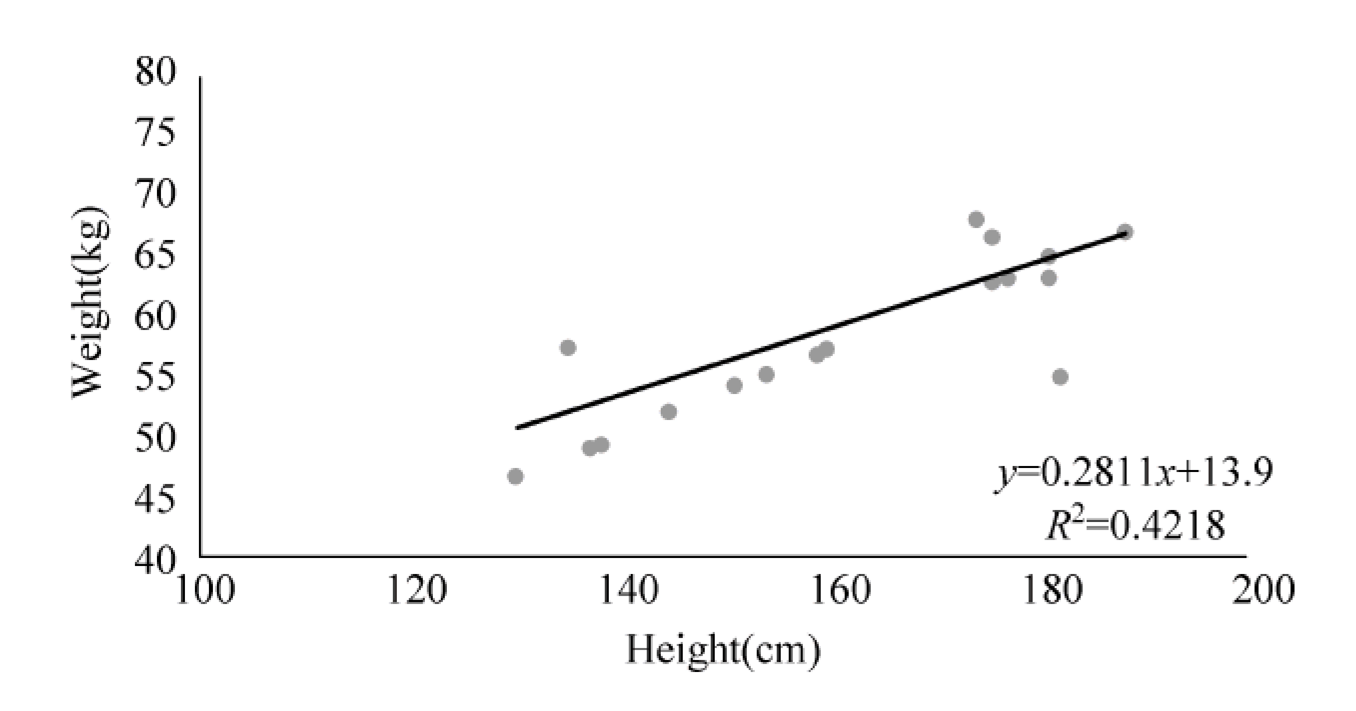


图 11-1　体重与身高的关系

我们找出最佳拟合直线 $y=0.2811x+13.9$。已知人的身高(height)，我们可以通过这条等式求出体重(weight)。

线性回归的两种主要类型是一元线性回归和多元线性回归。一元线性回归的特点是只有一个自变量。多元线性回归的特点正如其名，存在多个自变量。找最佳拟合直线的时候，你可以拟合到多项或者曲线回归。这些就被叫作多项或曲线回归。

Python 代码：

```
#Import Library
#Import other necessary libraries like Pandas, NumPy...
from sklearn import linear_model
#Load Train and Test datasets
#Identify feature and response variable(s) and values must be numeric and NumPy arrays
x_train = input_variables_values_training_datasets
y_train = target_variables_values_training_datasets
x_test = input_variables_values_test_datasets
# Create linear regression object
linear = linear_model.LinearRegression()
# Train the model using the training sets and check score
linear.fit(x_train, y_train)
linear.score(x_train, y_train)
#Equation coefficient and Intercept
```

```
print('Coefficient: n', linear.coef_)
print('Intercept: n', linear.intercept_)
#Predict Output
predicted = linear.predict(x_test)
```

11.2.2 逻辑回归

这是一个分类算法,而不是一个回归算法。该算法可根据已知的一系列因变量估计离散数值(如二进制数值 0 或 1 ,是或否,真或假)。简单来说,它通过将数据拟合进一个逻辑函数来预估一个事件出现的概率。因此,它也被叫作逻辑回归。因为它预估的是概率,所以它的输出值大小在 0 和 1 之间(正如所预计的一样)。如图 11-2 所示。

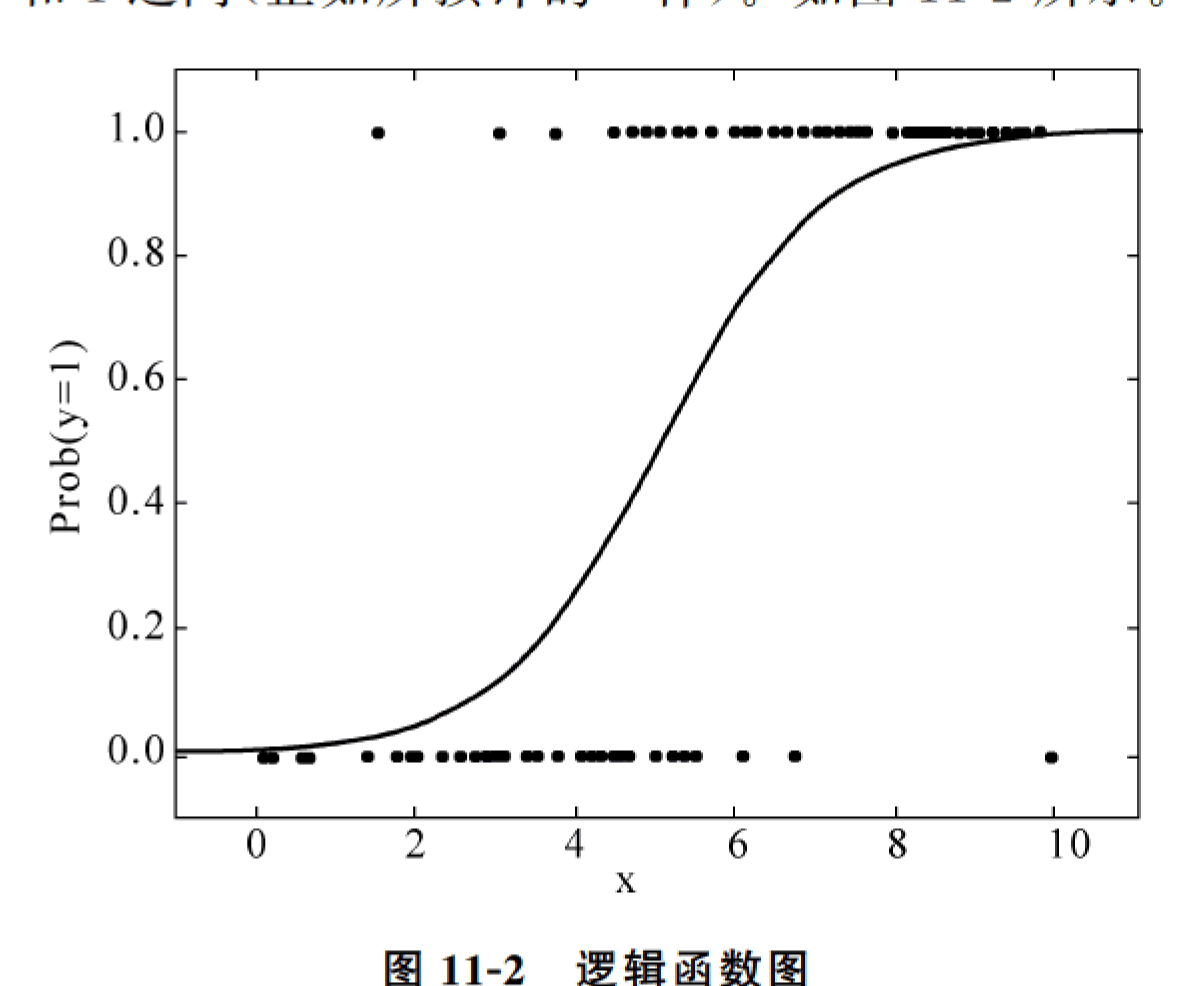


图 11-2 逻辑函数图

我们再通过一个简单的例子来理解这个算法。

假设你朋友让你解开一个谜题。只会有两个结果:你解开了或是没有解开。想象你要解答很多道题来找出你所擅长的主题。这个研究的结果就会像是这样:假设题目是一道十年级的三角函数题,你有 70%的可能会解开这道题。然而,若题目是道五年级的历史题,你只有 30%的可能性回答正确。这就是逻辑回归能提供给你的信息。

从数学上看,在结果中,概率的对数使用的是预测变量的线性组合模型。

```
odds = p/ (1 - p)  =  probability of event occurrence / probability of not event occurrence
ln(odds)  =  ln(p/(1 - p))
logit(p)  =  ln(p/(1 - p))  =  b0 + b1 * X1 + b2 * X2 + b3 * X3.... + bk * X
```

在上面的式子里,p 是我们感兴趣的特征出现的概率。它选用使观察样本值的可能性最大化的值作为参数,而不是通过计算误差平方和的最小值(就如一般的回归分析用到的一样)。

Python 代码:

```
#import Library
from sklearn.linear_model import LogisticRegression
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
```

```
# Create logistic regression object
model = LogisticRegression()
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Equation coefficient and Intercept
print('Coefficient: n', model.coef_)
print('Intercept: n', model.intercept_)
#Predict Output
predicted= model.predict(x_test)
```

更进一步：可以尝试更多的方法来改进这个模型：(1)加入交互项；(2)精简模型特性；(3)使用正则化方法；(4)使用非线性模型。

11.2.3　决策树

决策树这个监督式学习算法通常被用于分类问题。令人惊奇的是，它同时适用于分类变量和连续因变量。在这个算法中，我们将总体分成两个或更多的同类群。这是根据最重要的属性或者自变量来分成尽可能不同的组别。如图 11-3 所示。

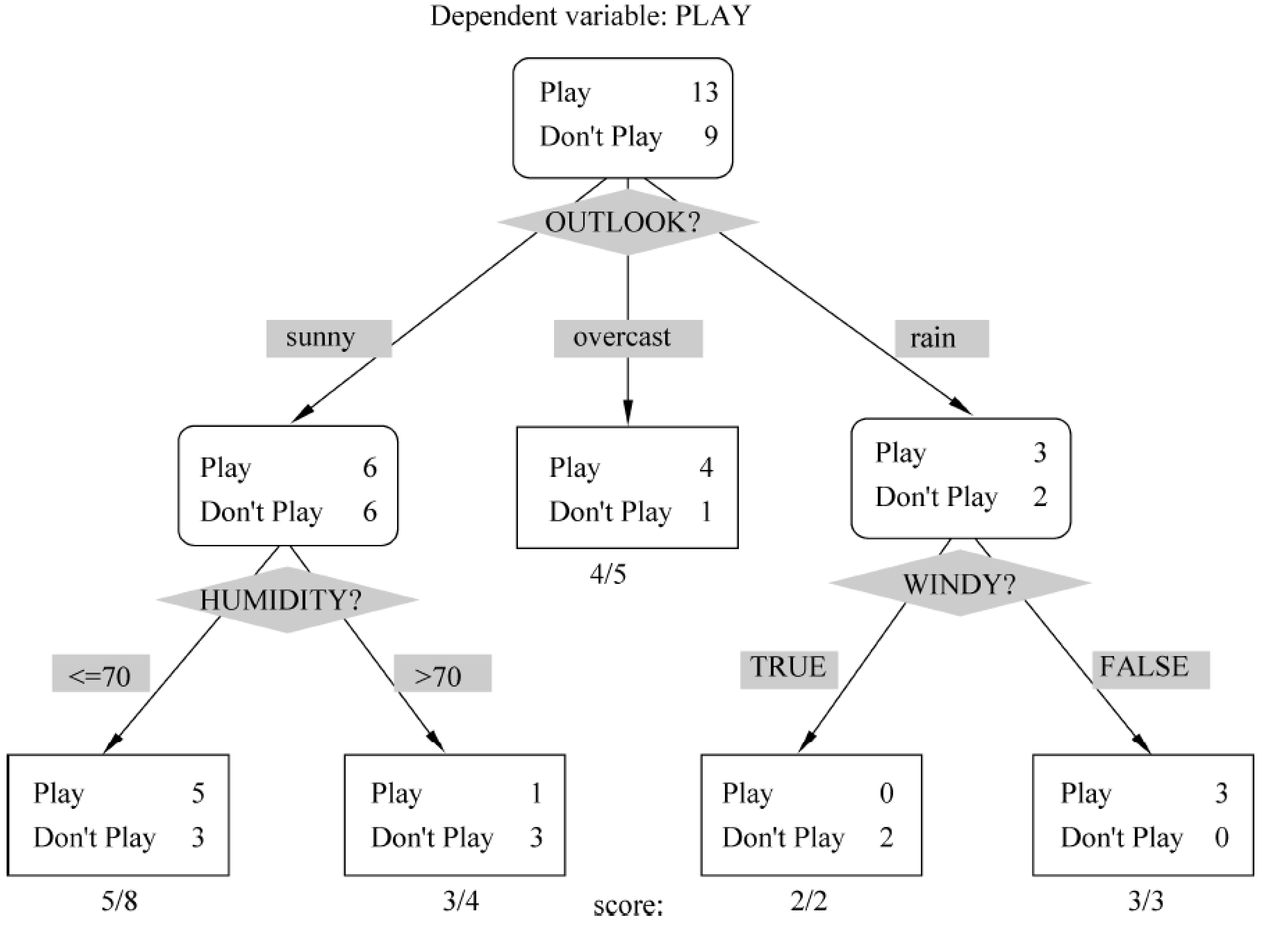


图 11-3　决策树

在图 11-3 中可以看到，根据多种属性，人群被分成了不同的四个小组，来判断"他们会不会去玩"。为了把总体分成不同组别，需要用到许多技术，比如说 Gini、Information Gain、Chi-square、entropy。

Python 代码：

```
#import Library
#import other necessary libraries like Pandas, NumPy...
```

```
from sklearn import tree
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
# Create tree object
model = tree.DecisionTreeClassifier(criterion = 'gini') # for classification, here you can
change the algorithm as gini or entropy (information gain) by default it is gini
# model = tree.DecisionTreeRegressor() for regression
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

11.2.4 支持向量机(SVM)分类

这是一种分类方法。在这个算法中,我们将每个数据在 N 维空间中用点标出(N 是所有的特征总数),每个特征的值是一个坐标的值。

例如,如果我们只有身高和头发长度两个特征,我们会在二维空间中标出这两个变量,每个点有两个坐标(这些坐标叫作支持向量),如图 11-4 所示。

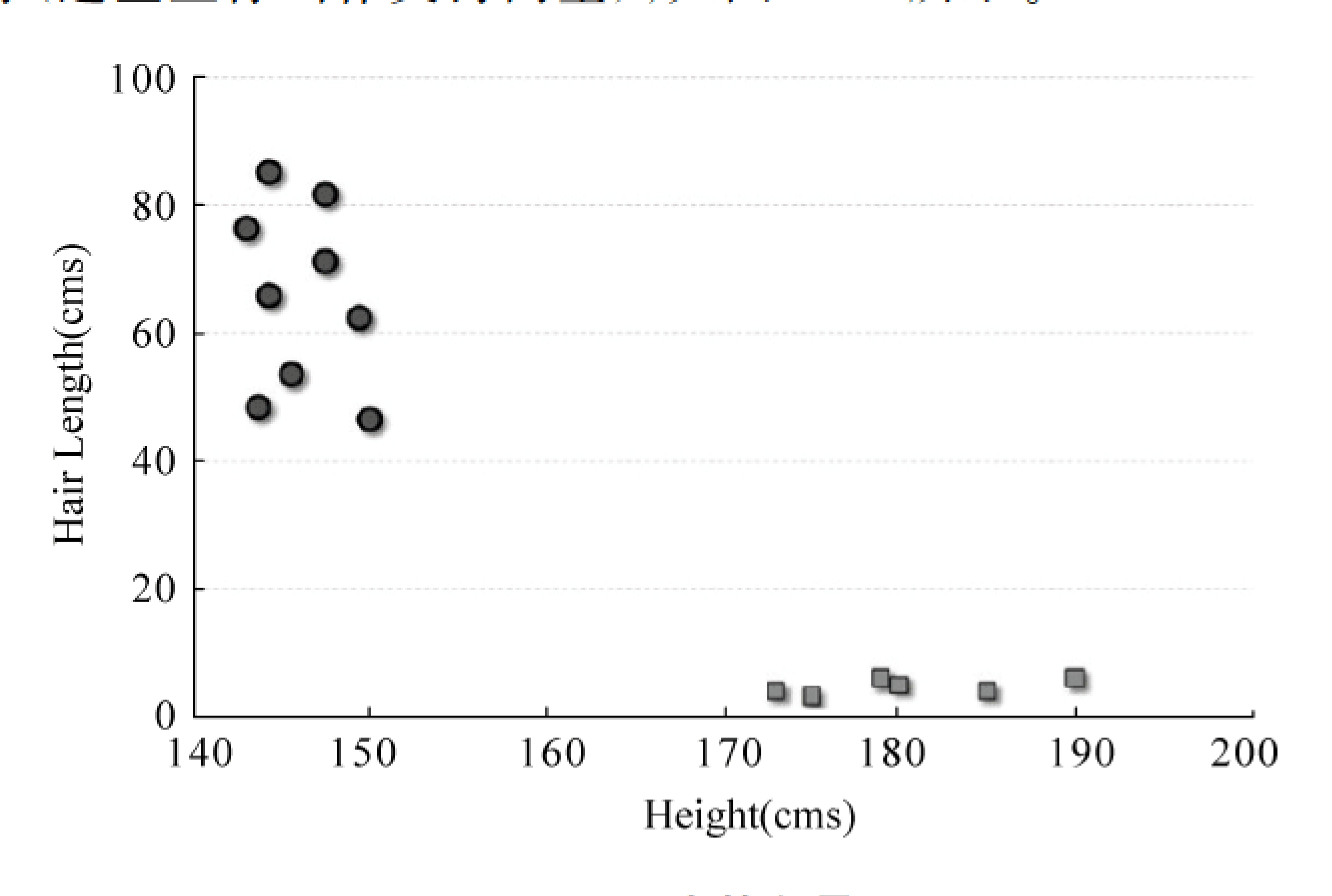


图 11-4 支持向量

现在,我们会找到将两组不同数据分开的一条直线。两个分组中距离最近的两个点到这条线的距离同时最优化,如图 11-5 所示。

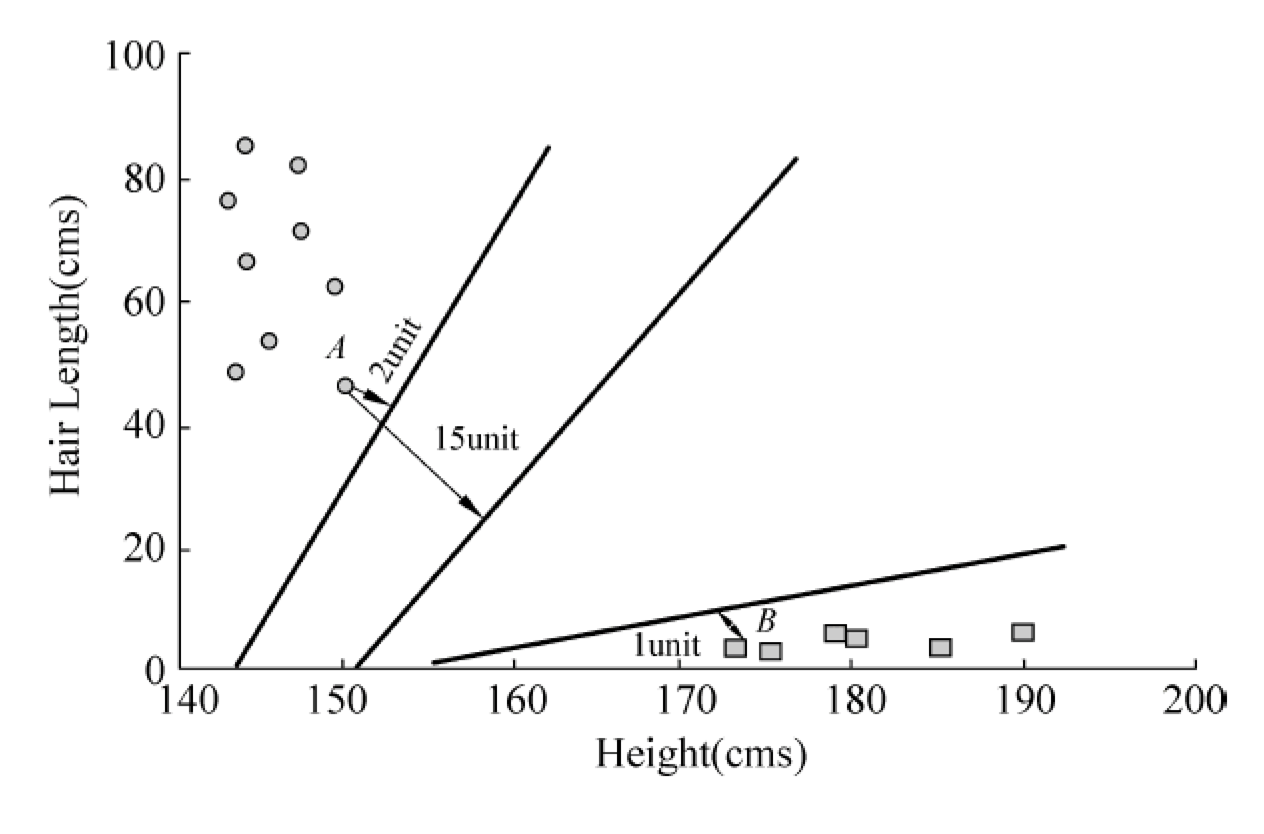


图 11-5 分开线

图 11-5 中的黑线将数据分类优化成两个小组，两组中距离最近的点（图中 A 点和 B 点）到达黑线的距离满足最优条件。这条直线就是我们的分割线。接下来，测试数据落到直线的哪一边，我们就将它分到哪一类去。

Python 代码：

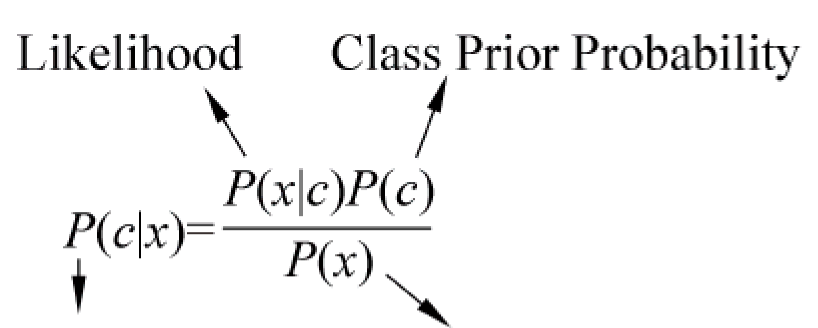
```
# Import Library
from sklearn import svm
# Assumed you have, X (predic
tor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create SVM classification object
model = svm. svc ( ) # there is various option associated with it, this is simple for
classification. You can refer link, for mo# re detail.
# Train the model using the training sets and check score
model. fit(X, y)
model. score(X, y)
# Predict Output
predicted = model. predict(x_test)
```

11.2.5　朴素贝叶斯分类

在预示变量间相互独立的前提下，根据贝叶斯定理可以得到朴素贝叶斯这个分类方法。用更简单的话来说，一个朴素贝叶斯分类器假设一个分类的特性与该分类的其他特性不相关。举例来说，如果一个水果又圆又红并且直径大约是 8cm，那么这个水果可能会是苹果。即便这些特性互相依赖或者依赖于别的特性存在，朴素贝叶斯分类器还是会假设这些特性分别独立地暗示这个水果是个苹果。

朴素贝叶斯模型易于建造，且对于大型数据集非常有用。虽然简单，但是朴素贝叶斯的表现却超越了非常复杂的分类方法。

贝叶斯定理提供了一种从 $P(c)$、$P(x)$ 和 $P(x|c)$ 计算后验概率 $P(c|x)$ 的方法。请看以下等式：



$$p(c \mid X) = P(x_1 \mid c) \cdot P(x_2 \mid c) \cdot \cdots \cdot P(x_n \mid c) \cdot P(c)$$

这里，$P(c|x)$ 是已知预示变量（属性）的前提下，类（目标）的后验概率，$P(c)$ 是类的先验概率，$P(x|c)$ 是可能性，即已知类的前提下，预示变量的概率，$P(x)$ 是预示变量的先验概率。

例 11-1　设有一个天气的训练集和对应的目标变量“Play”。我们需要根据天气情况，将“玩”和“不玩”的参与者进行分类。执行步骤如下。

步骤 1：把数据集转换成频率表。

步骤 2：利用类似“当 Overcast 可能性为 0.29 时，玩（Play）的可能性为 0.64”这样的概率，创造 Likelihood 表格。如图 11-6 所示。

步骤 3：使用朴素贝叶斯等式来计算每一类的后验概率。后验概率最大的类就是预测的结果。

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

图 11-6 表格

问题：如果天气晴朗，参与者就能玩。这个陈述正确吗？

我们可以使用讨论过的方法解决这个问题。于是 P(会玩|晴朗)＝P(晴朗|会玩)×P(会玩)/P(晴朗)

我们有 P(晴朗|会玩)＝3/9＝0.33，P(晴朗)＝5/14＝0.36，P(会玩)＝9/14＝0.64。

现在，P(会玩|晴朗)＝0.33×0.64/0.36＝0.60，有更大的概率。

朴素贝叶斯分类使用了一个相似的方法，通过不同属性来预测不同类别的概率。这个算法通常被用于文本分类，以及涉及多个类的问题。

Python 代码：

```
#Import Library
from sklearn.naive_bayes import GaussianNB
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
# Create SVM classification object model = GaussianNB() # there is other distribution for
multinomial classes like Bernoulli Naive Bayes, Refer link
# Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

11.2.6 K 最近邻(KNN)算法

K 最近邻算法(K-Nearest Neighbor)，又称 KNN 算法。该算法可用于分类问题和回归问题。然而，在业界内，K 最近邻算法更常用于分类问题。K 最近邻算法是一个简单的算法。它储存所有的案例，通过周围 K 个案例中的大多数情况划分新的案例。根据一个距离函数，新案例会被分配到它的 K 个近邻中最普遍的类别中去。

这些距离函数可以是欧式距离、曼哈顿距离、明氏距离或者是汉明距离。前三个距离函数用于连续函数，第四个距离函数(汉明函数)则被用于分类变量。如果 $K=1$，新案例就直接被分到离其最近的案例所属的类别中。有时候，使用 KNN 建模时，选择 K 的取值是一个挑战。如图 11-7 所示。

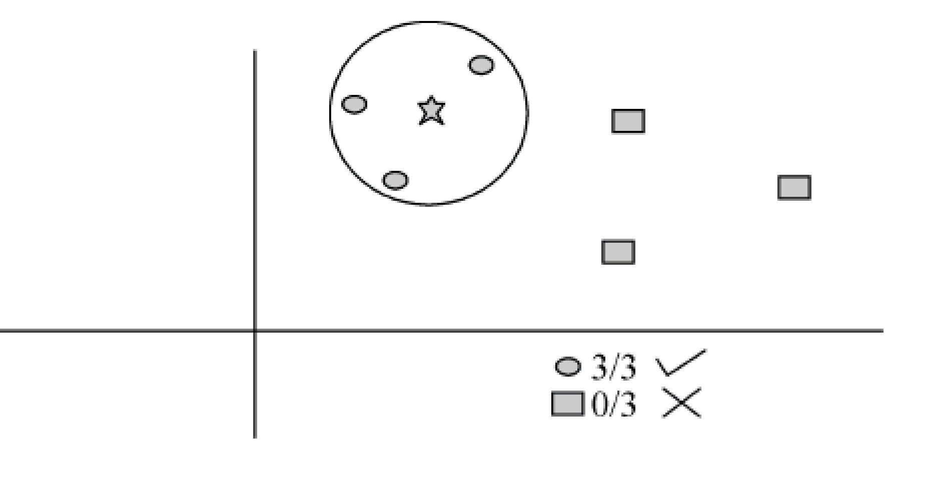


图 11-7 K 最近邻算法

在现实生活中广泛地应用了 K 最邻近算法。例如，想要了解一个完全陌生的人，你也许想要去找他的好朋友们或者他的圈子来获得他的信息。

在选择使用 K 最邻近算法之前，你需要考虑的事情：

Python 代码：

```
#Import Library
from sklearn.neighbors import KNeighborsClassifier
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
# Create KNeighbors classifier object model
KNeighborsClassifier(n_neighbors=6) # default value for n_neighbors is 5
# Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

11.2.7 K 均值算法

K 均值算法是一种非监督式学习算法，它能解决聚类问题。使用 K 均值算法来将一个数据归入一定数量的集群(假设有 K 个集群)的过程是简单的。一个集群内的数据点是均匀齐次的，并且异于别的集群。

K 均值算法形成集群的原理如下：

1) K 均值算法给每个集群选择 K 个点。这些点称作为质心。

2) 每一个数据点与距离最近的质心形成一个集群，也就是 K 个集群。

3) 根据现有的类别成员，找出每个类别的质心。现在我们有了新质心。

4) 当我们有新质心后，重复步骤 2 和步骤 3。找到距离每个数据点最近的质心，并与新的 k 集群联系起来。重复这个过程，直到数据都收敛了，也就是质心不再改变。

如何决定 K 值：

K 均值算法涉及集群，每个集群有自己的质心。一个集群内的质心和各数据点之间距离的平方和形成了这个集群的平方值之和。同时，当所有集群的平方值之和加起来的时候，就组成了集群方案的平方值之和。

我们知道，当集群的数量增加时，K 值会持续下降。但是，如果你将结果用图表来表示，你会看到距离的平方总和快速减少。到某个值 K 之后，减少的速度就大大下降了。在此，我们可以找到集群数量的最优值。如图 11-8 所示。

Python 代码：

```
#Import Library
from sklearn.cluster import KMeans
#Assumed you have, X (attributes) for training data set and x_test(attributes) of test_dataset
# Create KNeighbors classifier object model
k_means = KMeans(n_clusters=3, random_state=0)
# Train the model using the training sets and check score
model.fit(X)
#Predict Output
predicted= model.predict(x_test)
```

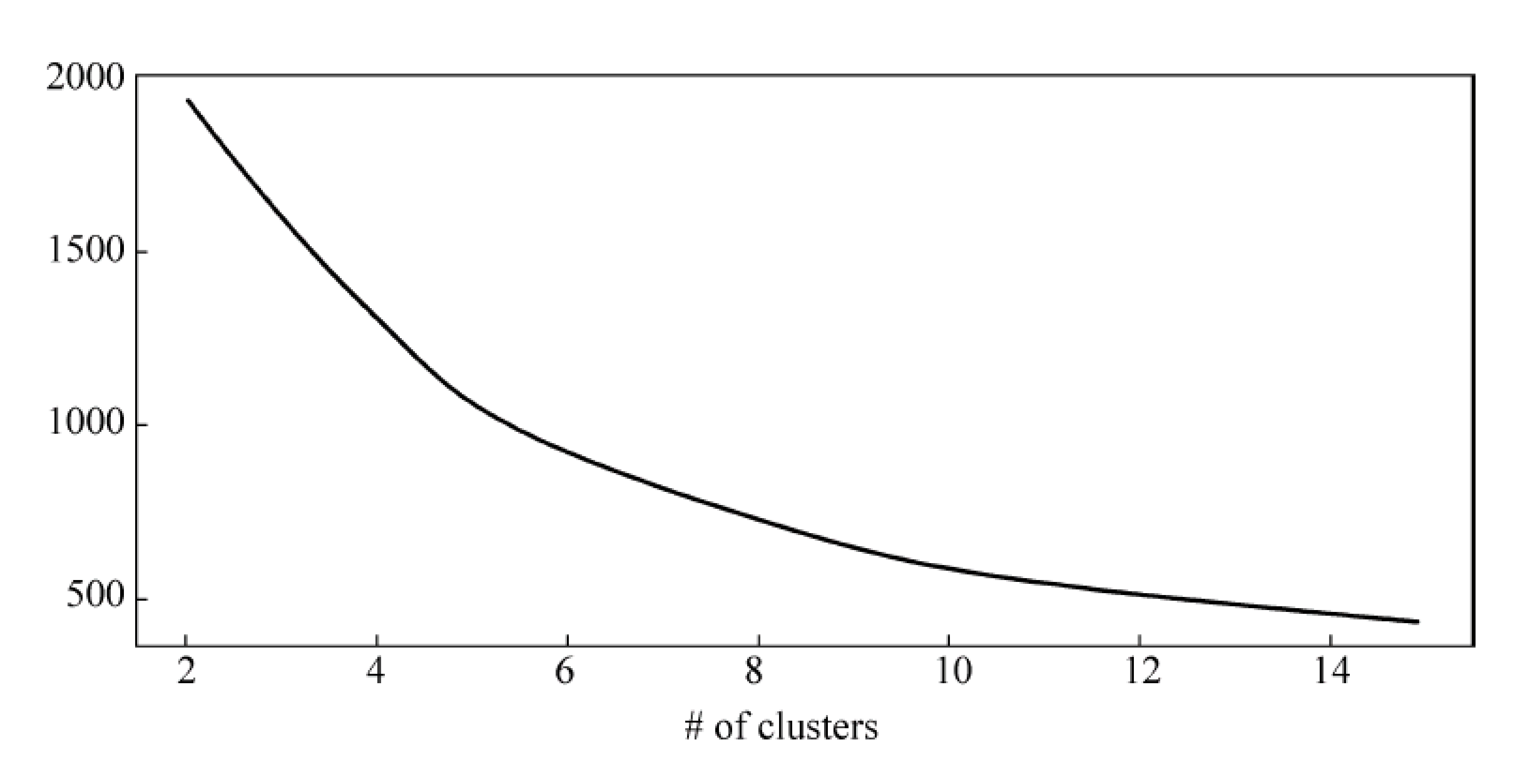


图 11-8　集群数量的最优值

11.2.8　随机森林算法

随机森林是表示决策树总体的一个专有名词。在随机森林算法中，我们有一系列的决策树（因此又名“森林”）。为了根据一个新对象的属性将其分类，每一个决策树有一个分类，称之为这个决策树“投票”给该分类。这个森林选择获得森林里（在所有树中）获得票数最多的分类。

每棵树是像这样种植养成的：

（1）如果训练集的案例数是 N，则从 N 个案例中用重置抽样法随机抽取样本。这个样本将作为“养育”树的训练集。

（2）假如有 M 个输入变量，则定义一个数字 $m<<M$。m 表示，从 M 中随机选中 m 个变量，这 m 个变量中最好的切分会被用来切分该节点。在种植森林的过程中，m 的值保持不变。

（3）尽可能大地种植每一棵树，全程不剪枝。

Python 代码：

```
#Import Library
from sklearn.ensemble import RandomForestClassifier
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
# Create Random Forest object
model= RandomForestClassifier()
# Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

11.2.9　降维算法

近年来，信息捕捉都呈指数增长。公司、政府机构、研究组织在应对着新资源以外，还要捕捉详尽的信息。

例如：电子商务公司更详细地捕捉关于顾客的资料：个人信息、网络浏览记录、他们的喜恶、购买记录、反馈以及别的许多信息，比你身边的杂货店售货员更加关注你。

作为一个数据科学家，我们提供的数据包含许多特点。这听起来给建立一个经得起考验的模型提供了很好的材料，但有一个挑战：如何从 1000 或者 2000 个变量中分辨出最重要的变量呢？在这种情况下，降维算法和别的一些算法（比如决策树、随机森林、PCA、因子分析）帮助我们根据相关矩阵、缺失的值的比例和别的要素来找出这些重要变量。

Python 代码：

```
#Import Library
from sklearn import decomposition
#Assumed you have training and test data set as train and test
# Create PCA obeject pca = decomposition.PCA(n_components = k) #default value of k = min(n_
sample, n_features)
# For Factor analysis
#fa = decomposition.FactorAnalysis()
# Reduced the dimension of training dataset using PCA
train_reduced = pca.fit_transform(train)
#Reduced the dimension of test dataset
test_reduced = pca.transform(test)
#For more detail on this, please refer??this link.
```

11.2.10　Gradient Boosting 和 AdaBoost 算法

当我们要处理很多数据来做一个有高预测能力的预测时，我们会用到 GBM 和 AdaBoost 这两种 boosting 算法。boosting 算法是一种集成学习算法。它结合了建立在多个基础估计值基础上的预测结果，来增进单个估计值的可靠程度。

Python 代码：

```
#Import Library
from sklearn.ensemble import GradientBoostingClassifier
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of
test_dataset
# Create Gradient Boosting Classifier object
model= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1,
random_state=0)
# Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

11.3　K 最近邻算法银行贷款分类的 Python 应用

K 最近邻分类算法是数据挖掘中较为简单的一种分类方法，通过计算不同数据点间的距离对数据进行分类，并对新的数据进行分类预测。前一节介绍过。本节介绍在 Python 中使用机器学习库 sklearn 建立 K 最近邻（K-Nearest Neighbor，KNN）模型的过程并使用模型对数据进行预测。

11.3.1　准备工作

首先是开始前的准备工作，导入需要使用的库文件。这里一共需要使用 5 个库文件。第一个是机器学习库，第二个是用于模型检验的交叉检验库，第三个是数值计算库，第四个

是科学计算库,最后是图表库。

```
#导入机器学习 KNN 分析库
from sklearn.neighbors import KNeighborsClassifier
#导入交叉验证库
from sklearn import cross_validation
#导入数值计算库
import NumPy as np
#导入科学计算库
import Pandas as pd
#导入图表库
import matplotlib.pyplot as plt
```

11.3.2 读取并查看数据表

读取并导入所需数据,创建名为 knn_data 的数据表(数据存在目录 G:\\2glkx\\data\\中),后面我们将使用这个数据对模型进行训练和检验。

```
#读取并创建名为 knn_data 的数据表
knn_data = pd.DataFrame(pd.read_csv('G:\\2glkx\\data\\knn_data.csv'))
```

使用 head 函数查看数据表的内容,这里只查看前 5 行的数据,数据表中包含三个字段,分别为贷款金额、用户收入和贷款状态。我们希望通过贷款金额和用户收入对最终的贷款状态进行分类和预测。

```
#查看数据表前 5 行
knn_data.head(5)
loan_amnt annual_inc loan_status
0     5000     54000     Fully Paid
1     2500     30000     Charged Off
2     2400     72252     Fully Paid
3    10000     89200     Fully Paid
4     5000     66000     Fully Paid
```

11.3.3 绘制散点图观察分类

创建模型之前先对数据汇总散点图,观察贷款金额和用户收入两个变量的关系以及对贷款状态的影响。下面是具体的代码,根据贷款状态将数据分为两组,第一组为 Fully Paid,第二组为 Charged Off。

```
#Fully Paid 数据集的 x1
fully_paid_loan = knn_data.loc[(knn_data["loan_status"] == "Fully Paid"),["loan_amnt"]]
#Fully Paid 数据集的 y1
fully_paid_annual = knn_data.loc[(knn_data["loan_status"] == "Fully Paid"),["annual_inc"]]
#Charge Off 数据集的 x2
charged_off_loan = knn_data.loc[(knn_data["loan_status"] == "Charged Off"),["loan_amnt"]]
#Charge Off 数据集的 y2
charged_off_annual = knn_data.loc[(knn_data["loan_status"] == "Charged Off"),["annual_
inc"]]
```

数据分组后开始绘制散点图，下面是绘图过程和具体的代码。

```
#设置图表字体为华文细黑,字号 15
plt.rc('font', family='STXihei', size=15)
#绘制散点图,Fully Paid 数据集贷款金额 x1,用户年收入 y1,设置颜色、标记点样式和透明度等参数
plt.scatter(fully_paid_loan,fully_paid_annual,color='#9b59b6',marker='^',s=60)
#绘制散点图,Charge Off 数据集贷款金额 x2,用户年收入 y2,设置颜色、标记点样式和透明度等参数
plt.scatter(charged_off_loan,charged_off_annual,color='#3498db',marker='o',s=60)
#添加图例,显示位置右上角
plt.legend(['Fully Paid', 'Charged Off'], loc='upper right')
#添加 x 轴标题
plt.xlabel('loan amount')
#添加 y 轴标题
plt.ylabel('annual_inc')
#添加图表标题
plt.title('loan amount&annual_inc')
#设置背景网格线颜色,样式,尺寸和透明度
plt.grid( linestyle='--', linewidth=0.2)
#显示图表
plt.show()
```

在散点图中，三角形为 Fully Paid 组，圆形为 Charged Off 组。我们所要做的是通过 KNN 模型对这两组数据的特征进行学习，例如从肉眼来看，Fully Paid 组用户收入要高于 Charged Off 组。并使用模型对新的数据进行分类预测。如图 11-9 所示。

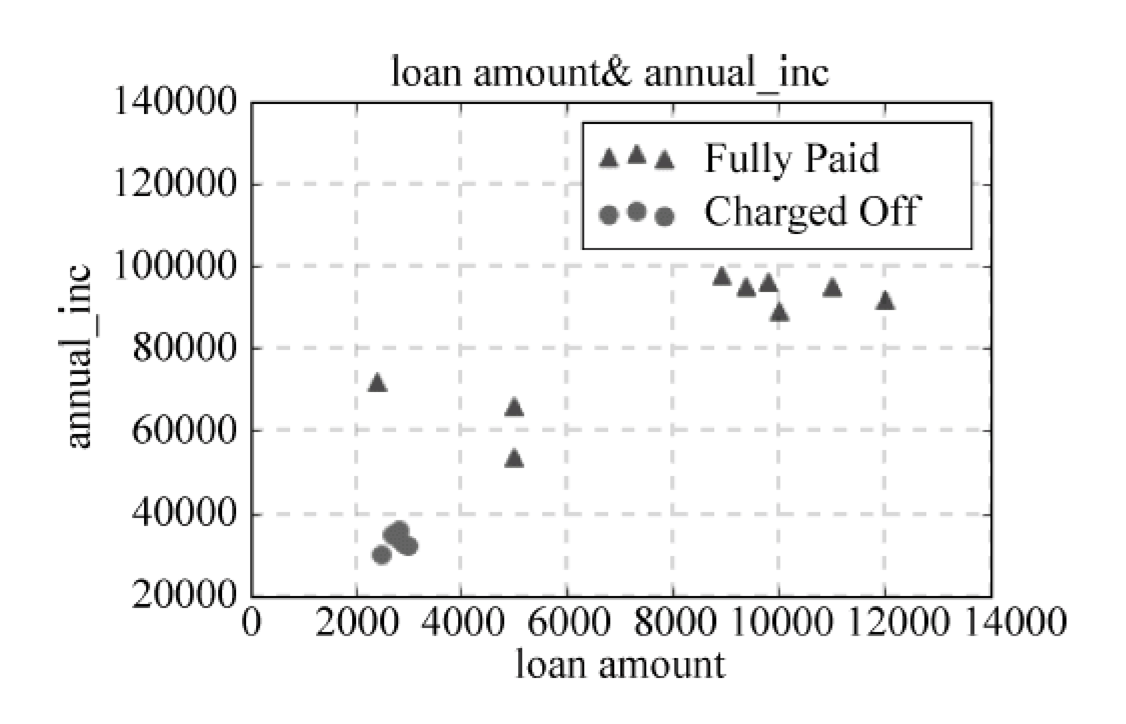


图 11-9　散点图

11.3.4　建立分类模型

1. 设置模型的自变量和因变量

创建 KNN 模型前，先设置模型中的自变量和因变量，也就是特征和分类。这里将贷款金额和用户收入设置为自变量，贷款状态是我们希望预测的结果，因此设置为因变量。

```
#将贷款金额和用户收入设为自变量 X
X = np.array(knn_data[['loan_amnt','annual_inc']])
#将贷款状态设为因变量 Y
Y = np.array(knn_data['loan_status'])
```

设置完成后查看自变量和因变量的行数，这里一共有 16 行数据，后面我们将把这 16 行数据分割为训练集和测试集，训练集用来建立模型，测试集则对模型的准确率进行检验。

```
#查看自变量和因变量的行数
X.shape,Y.shape
```

2. 将数据分割为训练集和测试集

采用随机抽样的方式将数据表分割为训练集和测试集，其中 60%的训练集数据用来训练模型，40%的测试集数据用来检验模型准确率。

```
#将原始数据通过随机方式分割为训练集和测试集,其中测试集占比为 40%
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, Y, test_size=0.4,
random_state=0)
```

分割后测试集的数据为 9 条。这些数据用来训练模型。

```
#查看训练集数据的行数
X_train.shape,y_train.shape
```

3. 对模型进行训练

将训练集数据 X_train 和 y_train 代入 KNN 模型中，对模型进行训练。下面是具体的代码和结果。

```
#将训练集代入 KNN 模型中
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train,y_train)
```

4. 使用测试集测对模型进行测试

使用测试集数据 X_test 和 y_test 对训练后的模型进行检验，模型准确率为 100%。

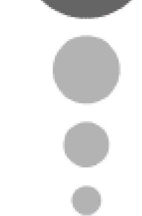
```
#使用测试集衡量模型准确度
clf.score(X_test, y_test)
```

完成训练和测试后，使用模型对新数据进行分类和预测。下面我们建立一组新的数据，贷款金额为 5000 元，用户收入为 40000 元，看看模型对新数据的分组结果。

```
#设置新数据,贷款金额 5000,用户收入 40000
new_data = np.array([[5000,40000]])
```

模型对新数据的分组结果为 Charged Off。这个分类准确吗？我们继续再来看下模型对这组新数据分组的概率。

```
#对新数据进行分类预测
clf.predict(new_data)
67%的概率为 Charged Off,33%的概率为 Fully Paid。根据这一概率模型将新数据划分到 Charged
Off 组。
#新数据属于每一个分类的概率
clf.classes_,clf.predict_proba(new_data)
```



```
(array(['Charged Off', 'Fully Paid'], dtype = object),
array([[ 0.66666667, 0.33333333]]))
```

11.4　各种机器学习算法的 Python 应用

11.4.1　载入数据

1. 需要准备的程序包

做机器学习应用,我们需要准备的程序包如下:

(1) NumPy,用于数据计算;

(2) SciPy,用于科学计算;

(3) Matplotlib,用于绘图;

(4) Scikit-learn,用于机器学习;

(5) IPython;集成系统。

2. 读取数据

首先我们需要载入一些用来操作的数据,使用的数据是非常简单的著名的花朵数据——安德森鸢尾花卉数据集。

我们有 150 个鸢尾花的一些尺寸的观测值:萼片长度、宽度,花瓣长度、宽度。还有它们的亚属:山鸢尾(iris setosa)、变色鸢尾(iris versicolor)和维吉尼亚鸢尾(iris Virginica)

向 Python 对象载入数据:

```
from sklearn import datasets
iris = datasets.load_iris()
```

数据存储在.data 项中,是一个(n_samples, n_features)数组。

```
iris.data.shape
Out[3]: (150, 4)
```

每个观察对象的种类存储在数据集的.target 属性中。这是一个长度为 n_samples 的整数一维数组:

```
iris.target.shape
Out[19]: (150,)
import NumPy as np
np.unique(iris.target)
Out[20]: array([0, 1, 2])
```

3. 一个改变数据集大小的示例:数码数据集(digits datasets)

数码数据集 1 包括 1797 个图像,每一个都是个代表手写数字的 8×8 像素图像。

```
digits = datasets.load_digits()
```

```
digits.images.shape
import pylab as pl
pl.imshow(digits.images[0], cmap = pl.cm.gray_r)
pl.show()
```

为了在 scikit 中使用这个数据集，我们把每个 8×8 图像转换成长度为 64 的矢量（或者直接用 digits. data）。

```
data = digits.images.reshape((digits.images.shape[0], -1))
```

4. 学习和预测

现在已经获得一些数据，要从中学习和预测一个新的数据。在 scikit-learn 中，我们通过创建一个估计器（estimator）从已经存在的数据学习，并且调用它的 fit(X, Y)方法。

```
from sklearn import svm
clf = svm.LinearSVC()
clf.fit(iris.data, iris.target) # learn from the data
```

得到如下结果：

```
Out[23]:
LinearSVC(C = 1.0, class_weight = None, dual = True, fit_intercept = True,
     intercept_scaling = 1, loss = 'squared_hinge', max_iter = 1000,
     multi_class = 'ovr', penalty = 'l2', random_state = None, tol = 0.0001,
     verbose = 0)
```

一旦我们已经从数据学习，我们可以使用我们的模型来预测未观测数据最可能的结果。

```
clf.predict([[ 5.0, 3.6, 1.3, 0.25]])
Out[24]: array([0])
```

注意：我们可以通过它以下划线结束的属性存取模型的参数。

```
clf.coef_
Out[25]:
array([[ 0.18423586, 0.45123243, -0.80794388, -0.45071334],
       [ 0.05107627, -0.89201609, 0.40574191, -0.9394283 ],
       [ -0.85076343, -0.98671944, 1.38098387, 1.8654622 ]])
```

11.4.2 分类

1. KNN 分类

(1) K 最近邻(KNN)分类器

最简单的可能的分类器是最近邻：给定一个新的观测值，将 n 维空间中最靠近它的训练样本标签给它。其中 n 是每个样本中的特性(features)数。

K 最近邻 2 分类器内部使用基于球树(ball tree)来代表它训练的样本。

KNN 分类示例：

```
# Create and fit a nearest-neighbor classifier
```

```
from sklearn import neighbors
from sklearn import neighbors
knn = neighbors.KNeighborsClassifier()
knn.fit(iris.data, iris.target)
Out[31]:
KNeighborsClassifier(algorithm = 'auto', leaf_size = 30, metric = 'minkowski',
           metric_params = None, n_jobs = 1, n_neighbors = 5, p = 2,
           weights = 'uniform')
knn.predict([[0.1, 0.2, 0.3, 0.4]])
Out[32]: array([0])
```

（2）训练集和测试集

当验证学习算法时，一定不要用一个用来拟合估计器的数据来验证估计器的预测。通过 KNN 估计器，我们将获得关于训练集完美的预测。

```
perm = np.random.permutation(iris.target.size)
iris.data = iris.data[perm]
iris.target = iris.target[perm]
knn.fit(iris.data[:100], iris.target[:100])
Out[33]:
KNeighborsClassifier(algorithm = 'auto', leaf_size = 30, metric = 'minkowski',
           metric_params = None, n_jobs = 1, n_neighbors = 5, p = 2,
           weights = 'uniform')
knn.score(iris.data[100:], iris.target[100:])
Out[34]: 1.0
```

3. 使用 LDA 模型进行分类

线性判别式分析（linear discriminant analysis，LDA），是模式识别的经典算法。通过对历史数据进行投影，以保证投影后同一类别的数据尽量靠近，不同类别的数据尽量分开。并生成线性判别模型，对新生成的数据进行分离和预测。本节使用机器学习库 scikit-learn 建立 LDA 模型，并通过绘图展示 LDA 的分类结果。

（1）准备工作

首先是开始前的准备工作，导入需要使用的库文件，本篇文章中除了常规的数值计算库 NumPy、科学计算库 Pandas 和绘图库 matplotlib 以外，还有绘图库中的颜色库，以及机器学习中的数据预处理和 LDA 库。

```
#导入数值计算库
import NumPy as np
#导入科学计算库
import Pandas as pd
#导入绘图库
import matplotlib.pyplot as plt
#导入绘图色彩库产生内置颜色
from matplotlib.colors import ListedColormap
#导入数据预处理库
from sklearn import preprocessing
#导入 linear discriminant analysis 库
```

```
from sklearn.lda import LDA
```

(2) 读取数据

读取并创建名称为 data 的数据表，后面我们将使用这个数据表创建 LDA 模型并绘图。

```
#读取数据并创建名为 data 的数据表
df = pd.read_csv('G:\\2glkx\\data\\pbdata1.csv')
data = pd.DataFrame(df)
```

使用 head 函数查看数据表的前 5 行，这里可以看到数据表共有三个字段，分别为贷款金额 loan_amnt、用户收入 annual_inc 和贷款状态 loan_status。

```
#查看数据表的前 5 行
data.head()
Out[8]:
   loan_amnt annual_inc loan_status
0       5000      54000  Fully Paid
1       2500      30000 Charged Off
2       2400      72252  Fully Paid
3      10000      89200  Fully Paid
4       5000      66000  Fully Paid
```

(3) 设置模型特征 X 和目标 Y

将数据表中的贷款金额和用户收入设置为模型特征 X，将贷款状态设置为模型目标 Y，也就是我们要分类的结果。

```
#设置贷款金额和用户收入为特征 X
X = np.array(data[['loan_amnt','annual_inc']])
#设置贷款状态为目标 Y
Y = np.array(data['loan_status'])
```

(4) 对特征进行标准化处理

贷款金额和用户收入间差异较大，属于两个不同量级的数据。因此需要对数据进行标准化处理，转化为无量纲的纯数值。

```
#特征数据进行标准化处理
scaler = preprocessing.StandardScaler().fit(X)
X_Standard = scaler.transform(X)
```

下面是经过标准化处理后的特征数据。

```
#查看标准化后的特征数据
X_Standard
Out[12]:
array([[-0.34202056, -0.44598557],
       [-1.05456341, -1.24831331],
       [-1.08306512,  0.16418467],
       [ 1.08306512,  0.73076178],
       [-0.34202056, -0.0448217 ],
       [-0.91205484, -1.18145267],
       [-0.96905827, -1.04773138],
```

```
        [ 1.6530994,    0.82436668],
        [ 1.36808226,   0.92465765],
        [-0.99755998,  -1.0811617 ],
        [ 1.02606169,   0.95808797],
        [ 0.91205484,   0.92465765],
        [ 0.79804798,   1.76041571],
        [-0.94055655,  -1.14802235],
        [-0.96905827,  -1.11459202],
        [ 0.76954627,   1.02494861]])
#设置分类平滑度
h = 0.01
```

(5) 创建 LDA 模型并拟合数据

将标准化后的特征 X 和目标 Y 代入 LDA 模型中。下面是具体的代码和计算结果。

```
#创建 LDA 模型
clf = LDA()
res = clf.fit(X_Standard,Y)
print clf
print res
```

得到如下结果：

```
LinearDiscriminantAnalysis(n_components = None, priors = None, shrinkage = None,
              solver = 'svd', store_covariance = False, tol = 0.0001)
LinearDiscriminantAnalysis(n_components = None, priors = None, shrinkage = None,
              solver = 'svd', store_covariance = False, tol = 0.0001)
```

11.4.3　分类支持向量机(SVM)

1. 线性支持向量机

分类支持向量机(SVM)尝试构建一个两个类别的最大间隔超平面。它选择输入的子集,调用支持向量即离分离的超平面最近的样本点。

```
from sklearn import svm
svc = svm.SVC(kernel = 'linear')
svc.fit(iris.data, iris.target)
```

得到如下结果：

```
Out[35]:
SVC(C = 1.0, cache_size = 200, class_weight = None, coef0 = 0.0,
  decision_function_shape = None, degree = 3, gamma = 'auto', kernel = 'linear',
  max_iter = -1, probability = False, random_state = None, shrinking = True,
  tol = 0.001, verbose = False)
```

scikit-learn 中有好几种支持向量机实现。最普遍使用的是 svm. SVC,svm. NuSVC 和 svm. LinearSVC;“SVC”代表支持向量分类器(Support Vector Classifier)(也存在回归 SVM,在 scikit-learn 中叫作“SVR”)。

练习：训练一个数字数据集的 svm. SVC。省略最后 10%并且检验观测值的预测表现。

2. 使用核

类别不总是可以用超平面分离，所以人们指望有些可能是多项式或指数实例的非线性决策函数。

(1) 线性核

```
svc = svm.SVC(kernel = 'linear')
```

(2) 多项式核

```
svc = svm.SVC(kernel = 'poly', ... degree = 3)
```

(3) RBF 核(径向基函数)

```
svc = svm.SVC(kernel = 'rbf')
# gamma: inverse of size of
# radial kernel
```

思考：上面的哪些核对数字数据集有更好的预测性能？(前两个)

11.4.4 聚类

给定鸢尾花数据集，如果知道这里有三种鸢尾花，但是无法得到它们的标签，可以尝试非监督学习：可以通过某些标准聚类观测值到几个组别里。

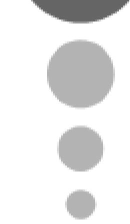
最简答的聚类算法是 K 均值算法。这将一个数据分成 K 个集群，以最小化观测值(n 维空间中)到聚类中心的均值来分配每个观测点到集群，然后均值重新被计算。这个操作递归运行直到聚类收敛，在 max_iter 回合内到最大值。

(一个替代的 K 均值算法实现在 SciPy 中的 cluster 包中。这个 scikit-learn 实现与之不同，通过提供对象 API 和几个额外的特性，包括智能初始化。)

```
from sklearn import cluster, datasets
iris = datasets.load_iris()
k_means = cluster.KMeans(3)
k_means.fit(iris.data)
Out[39]:
KMeans(copy_x = True, init = 'k - means++', max_iter = 300, n_clusters = 3, n_init = 10,
    n_jobs = 1, precompute_distances = 'auto', random_state = None, tol = 0.0001,
    verbose = 0)
print k_means.labels_[::10]
[1 1 1 1 1 2 2 2 2 2 0 0 0 0 0]
```

11.4.5 用主成分分析 PCA 降维

以上根据观测值标记的点，在一个方向非常平坦，所以一个特性几乎可以用其他两个确切地计算。PCA 可以发现哪个方向的数据不是平的，并且它可以通过在一个子空间投影来降维。注意：PCA 将在模块 decomposition 或 pca 中，这取决于 scikit-learn 的版本。




```
from sklearn import decomposition
pca = decomposition.PCA(n_components = 2)
pca.fit(iris.data)
Out[45]: PCA(copy = True, n_components = 2, whiten = False)
X = pca.transform(iris.data)
```

现在我们可以将(降维过的)鸢尾花数据集可视化：

```
import pylab as pl
pl.scatter(X[:, 0], X[:, 1], c = iris.target)
```

最后得到如图 11-10 所示的图形。

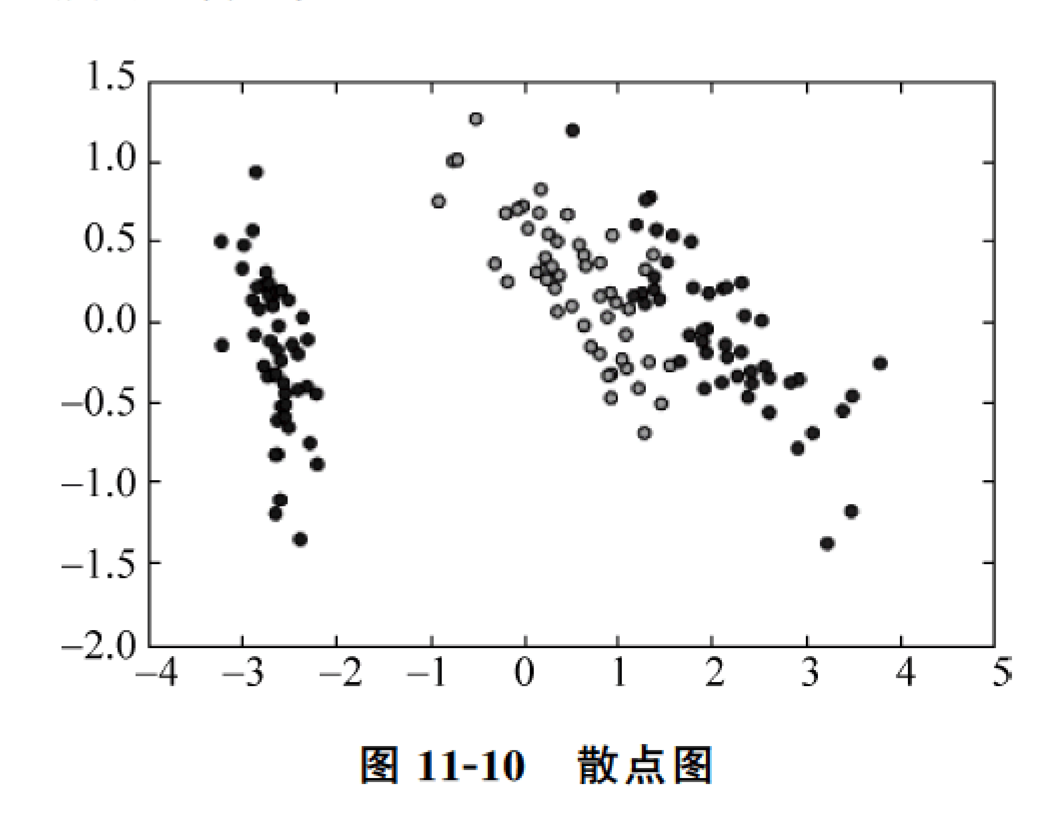


图 11-10　散点图

PCA 不仅在可视化高维数据集时非常有用。它可以用来作为帮助加速对高维数据不那么有效率的监督方法的预处理步骤。

11.4.6　线性模型：从回归到稀疏

1. 糖尿病数据集

糖尿病数据集包含测量 442 个病人而得的 10 项生理指标(年龄、性别、体重、血压)，和一年后疾病进展的指示：

```
diabetes = datasets.load_diabetes()
diabetes_X_train = diabetes.data[: - 20]
diabetes_X_test = diabetes.data[ - 20:]
diabetes_y_train = diabetes.target[: - 20]
diabetes_y_test = diabetes.target[ - 20:]
```

我们的任务是从生理指标预测疾病。

2. 稀疏模型

为了改善问题的条件(无信息变量，减少维度的不利影响，作为一个特性(feature)选择的预处理，等等)，我们只关注有信息的特性，将没有信息的特性设置为 0。这个罚则函数法，叫作套索(lasso)，可以将一些系数设置为 0。这些方法叫作稀疏方法(sparse method)，稀疏化可以被视作奥卡姆剃刀：相对于复杂模型更倾向于简单的。

```
from sklearn import linear_model
regr = linear_model.Lasso(alpha = .3)
regr.fit(diabetes_X_train, diabetes_y_train)
Out[54]:
Lasso(alpha = 0.3, copy_X = True, fit_intercept = True, max_iter = 1000,
   normalize = False, positive = False, precompute = False, random_state = None,
   selection = 'cyclic', tol = 0.0001, warm_start = False)
regr.coef_ # very sparse coefficients
Out[55]:
array([    0.          ,    -0.         ,   497.34075682, 199.17441034,
         -0.          ,    -0.         , -118.89291545,    0.         ,
        430.9379595   ,     0.         ])
regr.score(diabetes_X_test, diabetes_y_test)
Out[56]: 0.55108354530029768
```

这个分数和线性回归(最小二乘法)非常相似：

```
lin = linear_model.LinearRegression()
lin.fit(diabetes_X_train, diabetes_y_train)
```

得到如下结果：

```
Out[65]: LinearRegression(copy_X = True, fit_intercept = True, n_jobs = 1, normalize = False)
lin.score(diabetes_X_test, diabetes_y_test)
Out[66]: 0.58507530226905735
```

3. 同一问题的不同算法

同一数学问题可以用不同算法解决。例如，sklearn 中的 lasso 对象使用坐标下降(coordinate descent)方法解决套索回归，这在大数据集时非常有效率。然而，sklearn 也提供了 lasso LARS 对象，LARS 在解决权重向量估计非常稀疏、观测值很少的问题时是很有效率的方法。

11.4.7 交叉验证估计器

交叉验证在一个 algorithm by algorithm 基础上可以更有效地设定参数。这就是为什么对给定的估计器，scikit-learn 使用“CV”估计器，通过交叉验证自动设定参数。

```
from sklearn import linear_model, datasets
lasso = linear_model.LassoCV()
diabetes = datasets.load_diabetes()
X_diabetes = diabetes.data
y_diabetes = diabetes.target
lasso.fit(X_diabetes, y_diabetes)
Out[67]:
LassoCV(alphas = None, copy_X = True, cv = None, eps = 0.001, fit_intercept = True,
    max_iter = 1000, n_alphas = 100, n_jobs = 1, normalize = False, positive = False,
    precompute = 'auto', random_state = None, selection = 'cyclic', tol = 0.0001,
    verbose = False)
# The estimator chose automatically its lambda:
```

```
lasso.alpha
Out[68]: 0.013180196198701137
```

这些估计器是相似的,以‘CV’为它们名字的后缀。

11.5　K 最近邻算法分类的 Python 应用

11.5.1　人工生成数据的 K 最近邻法分类 Python 应用

scikit-learn 是 Python 中一个功能非常齐全的机器学习库,本节将介绍如何用 scikit-learn 来进行 KNN 分类计算。

准备工作如下:

```
from sklearn import neighbors
```

1. 初始化及其功能解释

我们讨论的是 scikit-learn 库中的 neighbors. KNeighborsClassifier,翻译为 K 最近邻分类功能,也就是我们常说的 KNN(K-Nearest Neighbors)。

首先进行这个类初始化:

```
neighbors.KNeighborsClassifier(n_neighbors = 5, weights = 'uniform', algorithm = 'auto', leaf_
size = 30, p = 2, metric = 'minkowski', metric_params = None, n - jobs = 1)。
```

其中:

(1) n_neighbors: 是 KNN 里的 K,就是在做分类时,我们选取问题点最近的多少个最近邻;

(2) weights 是在进行分类判断时给最近邻附上的加权,默认的'uniform'是等权加权,还有'distance'选项是按照距离的倒数进行加权,也可使用用户自己设置的其他加权方法。例如: 假如距离询问点最近的三个数据点中,有 1 个 A 类和 2 个 B 类,并且假设 A 类离询问点非常近,而两个 B 类距离则稍远。在等权加权中,选取问题点最近邻的行会判断问题点为 B 类; 而如果使用距离加权,那么 A 类有更高的权重(因为更近),如果它的权重高于两个 B 类的权重的总和,那么算法会判断问题点为 A 类。权重功能的选项应该视应用的场景而定。

(3) algorithm 是分类时采取的算法,有'brute'、'kd_tree'和'ball_tree'。kd_tree 的算法在 kd 树中有详细介绍,而 ball_tree 是另一种基于树状结构的 kNN 算法,brute 则是最直接的蛮力计算。根据样本量的大小和特征的维度数量,不同的算法有各自的优势。默认的'auto'选项会在学习时自动选择最合适的算法,所以一般来讲选择 auto 即可。

(4) leaf_size 是 kd_tree 或 ball_tree 生成的树的树叶(树叶就是二叉树中没有分枝的节点)的大小。在 kd 树中我们所有的二叉树的叶子中都只有一个数据点,但实际上树叶中可以有多于一个的数据点,算法在达到叶子时在其中执行蛮力计算即可。对于很多使用场景来说,叶子的大小并不是很重要,我们设 leaf_size=1 就好。

(5) metric 和 p,是 KNN 中的距离函数的选项,如果 metric ='minkowski'并且 p=p 的话,计算两点之间的距离就是

$$d((x_1,\cdots,x_n),(y_1,\cdots,y_n)) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

一般来讲，默认的 metric='minkowski'(默认)和 p=2(默认)就可以满足大部分需求。其他的 metric 选项可参见相关说明文档。

(6) metric_params 是一些特殊 metric 选项需要的特定参数，默认是 None。

(7) n_jobs 是并行计算的线程数量，默认是 1，输入 -1 则设为 CPU 的内核数。

2. 拟合及其功能解释

在创建了一个 KNeighborsClassifier 类之后，我们需要通过数据来进行学习。这时需要使用 fit()拟合功能。

```
neighbors.KNeighborsClassifier.fit(X,y)
```

在这里：X 是一个 list 或 array 的数据，每一组数据可以是 tuple 也可以是 list 或者一维 array，但要注意所有数据的长度必须一样(等同于特征的数量)。当然，也可以把 X 理解为一个矩阵，其中每一横行是一个样本的特征数据。y 是一个和 X 长度相同的 list 或 array，其中每个元素是 X 中相对应的数据的分类标签。

KNeighborsClassifier 类在对训练数据执行 fit()后会根据原先 algorithm 的选项，依据训练数据生成一个 kd_tree 或者 ball_tree。如果输入是 algorithm='brute'，则什么都不做。这些信息都会被保存在一个类中，我们可以用它进行预测和计算。几个常用的功能如下。

(1) K 最近邻

```
neighbors.KNeighborsClassifier.kneighbors(X = None, n_neighbors = None, return_distance =
True)
```

这里 X 是一个 list 或 array 的坐标，如果不提供，则默认输入训练时的样本数据。

n_neighbors 是指定搜寻最近的样本数据的数量，如果不提供，则以初始化 kNeighborsClassifier 时的 n_neighbors 为准。

这个功能输出的结果是(dist=array[array[float]]，index=array[array[int]])。index 的长度和 X 相同，index[i]是长度为 n_neighbors 的一 array 的整数；假设训练数据是 fit(X_train，y_train)，那么 X_train(index[i][j])是在训练数据(X_train)中离 X[i]第 j 近的元素，并且 dist[i][j]是它们之间的距离。

输入的 return_distance 是是否输出距离，如果选择 False，那么功能的输出会只有 index 而没有 dist。

(2) 预测

```
neighbors.kNeighborsClassifier.predict(X)
```

也许是最常用的预测功能。输入 X 是一个 list 或 array 的坐标，输出 y 是一个长度相同的 array，y[i]是通过 KNN 分类对 X[i]所预测的分类标签。

(3) 概率预测

```
neighbors.kNeighborsClassifier.predict_proba(X)
```

输入和上面的相同，输出 p 是 array[array[float]]，p[i][j]是通过概率 KNN 判断 X[i]属于第 j 类的概率。这里类别的排序是按照词典排序。举例来说，如果训练用的分类标签里有

(1,'1','a')三种,那么 1 就是第 0 类,'1'是第 1 类,'a'是第 2 类,因为在 Python 中 1<'1'<'a'。

(4) 正确率打分

```
neighbors.KNeighborsClassifier.score(X, y, sample_weight = None)
```

这是用来评估一次 KNN 学习的准确率的方法。很多可能会因为样本特征的选择不当或者 K 值的选择不当而出现过拟合或者偏差过大的问题。为了保证训练方法的准确性,一般我们会将已经带有分类标签的样本数据分成两组,一组进行学习,一组进行测试。这个 score()就是在学习之后进行测试的功能。同 fit()一样,这里的 X 是特征坐标,y 是样本的分类标签;sample_weight 是对样本的加权,长度等于 sample 的数量。返回的是正确率的百分比。

3. 人工生成数据 KNN 分类例子

除了 sklearn.neighbors,还需要导入 NumPy 和 matplotlib 画图。

```
import random
from sklearn import neighbors
import NumPy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

我们随机生成 6 组 200 个的正态分布:

```
x1  = np.random.normal(50, 6, 200)
y1  = np.random.normal(5, 0.5, 200)
x2  = np.random.normal(30,6,200)
y2  = np.random.normal(4,0.5,200)
x3  = np.random.normal(45,6,200)
y3  = np.random.normal(2.5, 0.5, 200)
```

x1、x2、x3 作为 x 坐标,y1、y2、y3 作为 y 坐标,两两配对。(x1,y1)标为 1 类,(x2,y2)标为 2 类,(x3,y3)是 3 类。将它们画出得到下图,1 类是(蓝色)矩形,2 类(红色)三角形,3 类(绿色)圆形。

```
plt.scatter(x1,y1,c = 'b',marker = 's',s = 50,alpha = 0.8)
plt.scatter(x2,y2,c = 'r', marker = '^', s = 50, alpha = 0.8)
plt.scatter(x3,y3, c = 'g', s = 50, alpha = 0.8)
```

得到如图 11-11 所示的图形。

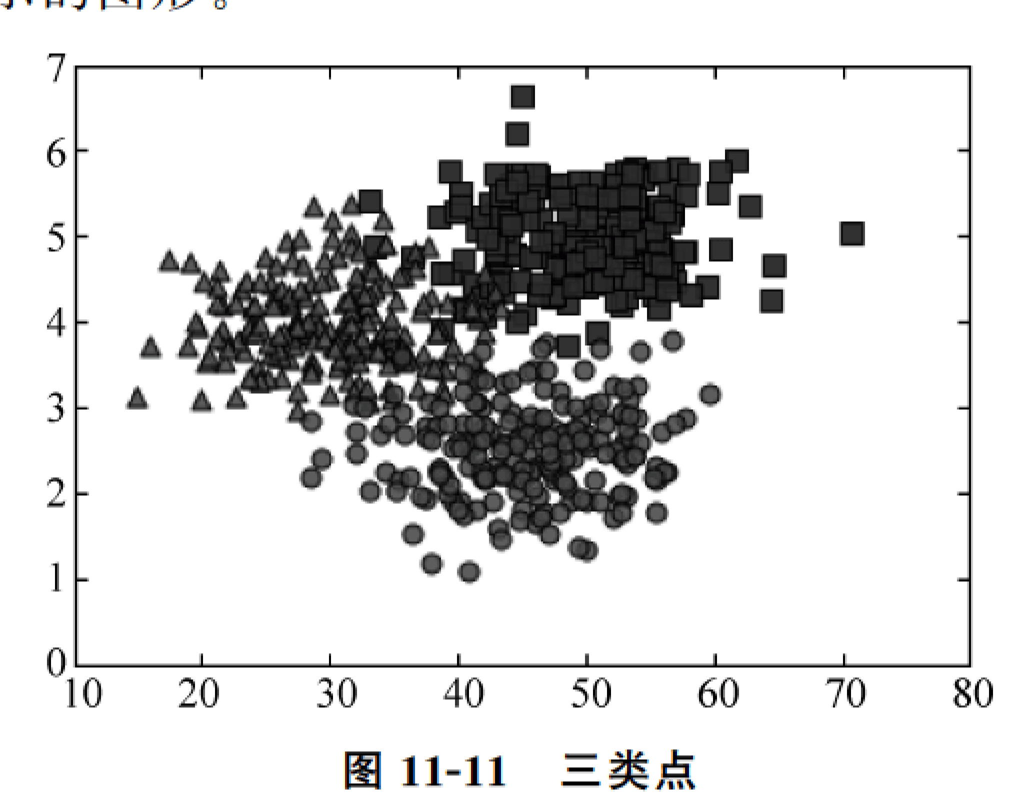


图 11-11　三类点

把所有的 x 坐标和 y 坐标放在一起。

```
x_val = np.concatenate((x1,x2,x3))
y_val = np.concatenate((y1,y2,y3))
```

求出 x 值的最大差和 y 值的最大差。

```
x_diff = max(x_val) - min(x_val)
y_diff = max(y_val) - min(y_val)
```

将坐标除以这个差以归一化，再将 x 和 y 值两两配对。

```
x_normalized = [x/(x_diff) for x in x_val]
y_normalized = [y/(y_diff) for y in y_val]
xy_normalized = zip(x_normalized,y_normalized)
```

这样我们就准备好了训练使用的特征数据，还需要生成相应的分类标签。生成一个长度 600 的 list，前 200 个是 1，中间 200 个是 2，最后 200 个是 3，对应三种标签。

```
labels = [1] * 200 + [2] * 200 + [3] * 200
```

然后，就要生成 sklearn 的 K 最近邻分类功能了。参数中，n_neighbors 设为 30，其他的都使用默认值即可。

```
clf = neighbors.KNeighborsClassifier(30)
```

（注意：我们是从 sklearn 里导入了 neighbors。如果是直接导入了 sklearn，应该输入 sklearn. neighbors. KNeighborsClassifier()）

下面进行拟合。归一化的数据是 xy_normalized，分类标签是 labels。

```
bors.KNeighborsClassifier()
```

就这么简单。下面我们来实现一些功能。

（1）K 最近邻

首先，我们想知道(50,5)和(30,3)两个点附近最近的 5 个样本分别都是什么。注意，坐标别忘了除以 x_diff 和 y_diff 来归一化。

```
nearests = clf.kneighbors([(50/x_diff, 5/y_diff),(30/x_diff, 3/y_diff)], 5, False)
nearests
```

得到

```
array([[ 33, 64, 122, 52, 53],[200, 460, 505, 294, 490]])
```

也就是说训练数据中的第 33,64,122,52,53 个离(50,5)最近，第 200,460,505,294,490 个离(30,3)最近。

（2）预测

还是上面那两个点，我们通过 30NN 来判断它们属于什么类别。

```
prediction = clf.predict([(50/x_diff, 5/y_diff),(30/x_diff, 3/y_diff)])
prediction
```

得到

```
array([1, 2])
```

也就是说(50,5)判断为 1 类,而(30,3)是 2 类。

(3) 概率预测

那么这两个点的分类的概率都是多少呢?

```
prediction_proba = clf.predict_proba([(50/x_diff,5/y_diff),(30/x_diff,3/y_diff)])
prediction_proba
```

得到

```
array([[ 1.        ,  0.        , 0.        ],
       [ 0.        ,  0.66666667, 0.33333333]])
```

这个结果告诉我们,(50, 5)有 100%的可能性是 1 类,而(30,3)有 67%是 2 类,33%是 3 类。

(4) 准确率打分

我们再用同样的均值和标准差生成一些正态分布点,以此检测预测的准确性。

```
x1_test  = np.random.normal(50, 6, 100)
y1_test  = np.random.normal(5, 0.5, 100)
x2_test  = np.random.normal(30,6,100)
y2_test  = np.random.normal(4,0.5,100)
x3_test  = np.random.normal(45,6,100)
y3_test  = np.random.normal(2.5, 0.5, 100)
xy_test_normalized  =  zip(np.concatenate((x1_test,x2_test,x3_test))/x_diff,\
                           np.concatenate((y1_test,y2_test,y3_test))/y_diff)
labels_test  =  [1] * 100 + [2] * 100 + [3] * 100
```

测试数据生成完毕,下面进行测试。

```
score  =  clf.score(xy_test_normalized, labels_test)
Score
0.9666666666666667
```

可见我们得到预测的正确率是 97%,还是很不错的。

再看一下,如果使用 1NN 分类,会出现过拟合的现象,那么准确率的评分就变为:

```
clf1  =  neighbors.KNeighborsClassifier(1)
clf1.fit(xy_normalized, labels)
clf1.score(xy_test_normalized, labels_test)
0.9533333333333337
```

我们得到 95%的正确率,的确是降低了。我们还应该注意,这里的预测准确率很高是因为训练和测试的数据都是人为按照正态分布生成的,在实际使用的很多场景中(比如,涨跌预测)是很难达到这个精度的。

（5）生成些漂亮的图

KNN 分类图，一般只能展示两个维度的数据，超过三个特征就画不出来了。

首先我们需要生成一个区域里大量的坐标点。这要用到 np.meshgrid()函数。给定两个 array，比如 x=[1,2,3]和 y=[4,5]，np.meshgrid(x,y)会输出两个矩阵

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

和

$$\begin{bmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \end{bmatrix}$$

这两个叠加到一起得到 6 个坐标：

$$\begin{bmatrix} (1,4) & (2,4) & (3,4) \\ (1,5) & (2,5) & (3,5) \end{bmatrix}$$

就是以[1,2,3]为横轴，[4,5]为竖轴所得到的长方形区间内的所有坐标点。我们现在要生成[1,80]×[1,7]的区间里的坐标点，横轴要每 0.1 一跳，竖轴每 0.01 一跳。于是

```
xx,yy = np.meshgrid(np.arange(1,70.1,0.1), np.arange(1,7.01,0.01))
```

于是 xx 和 yy 都是 601×691 的矩阵。还有，不要忘了除以 x_diff 和 y_diff 来将坐标归一化。

```
xx_normalized = xx/x_diff
yy_normalized = yy/y_diff
```

下面，np.ndarray.ravel()功能可以把一个矩阵抻直成一个一维 array，把

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

变成

$$[1\ 2\ 3\ 1\ 2\ 3]$$

np.c_()又把两个 array 粘起来(类似于 zip)，输入

$$[1\ 2\ 3\ 1\ 2\ 3]$$

和

$$[4\ 4\ 4\ 5\ 5\ 5]$$

输出

$$\begin{bmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 4 & 4 & 4 & 5 & 5 & 5 \end{bmatrix}$$

或者理解为

```
{(1,4),(2,4),(3,4),(1,5),(2,5),(3,5)}{(1,4),(2,4),(3,4),(1,5),(2,5),(3,5)}
```

于是

```
coords = np.c_[xx_normalized.ravel(), yy_normalized.ravel()]
```

得到一个 array 的坐标。下面就可以进行预测：

```
Z = clf.predict(coords)
```

当然，Z 是一个一维 array，为了和 xx 还有 yy 相对应，要把 Z 的形状再转换回矩阵。

```
Z = Z.reshape(xx.shape)
```

下面用 pcolormesh 画出背景颜色。这里，ListedColormap 是自己生成 colormap 的功能，#rrggbb 颜色的 rgb 代码。pcolormesh 会根据 Z 的值(1、2、3)选择 colormap 里相对应的颜色。

```
light_rgb = ListedColormap([ '#AAAAFF', '#FFAAAA','#AAFFAA'])
plt.pcolormesh(xx, yy,Z, cmap = light_rgb)
plt.scatter(x1,y1,c = 'b',marker = 's',s = 50,alpha = 0.8)
plt.scatter(x2,y2,c = 'r', marker = '^', s = 50, alpha = 0.8)
plt.scatter(x3,y3, c = 'g', s = 50, alpha = 0.8)
plt.axis((10, 70,1,7))
```

得到如图 11-12 所示的图形。

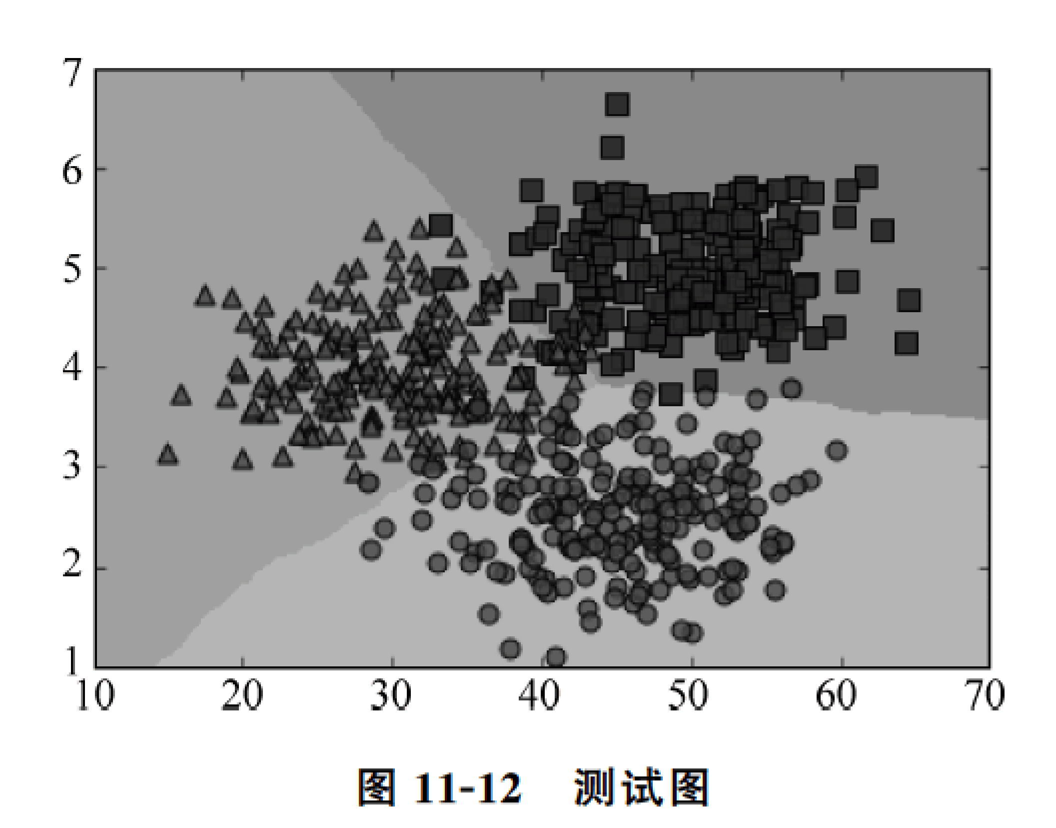


图 11-12　测试图

下面进行概率预测，使用

```
z_proba = clf.predict_proba(coords)
```

得到每个坐标点的分类概率值。假设我们想画出红色(三角形)的概率，那么提取所有坐标的 2 类概率，转换成蓝色矩阵形状。

```
z_proba_reds = z_proba[:,1].reshape(xx.shape)
```

再选一个预设好的红色调 cmap 画出来

```
plt.pcolormesh(xx, yy,Z_proba_reds, cmap = 'Reds')
plt.scatter(x1,y1,c = 'b',marker = 's',s = 50,alpha = 0.8)
plt.scatter(x2,y2,c = 'r', marker = '^', s = 50, alpha = 0.8)
plt.scatter(x3,y3, c = 'g', s = 50, alpha = 0.8)
plt.axis((10, 70,1,7))
```

得到如图 11-13 所示的图形。

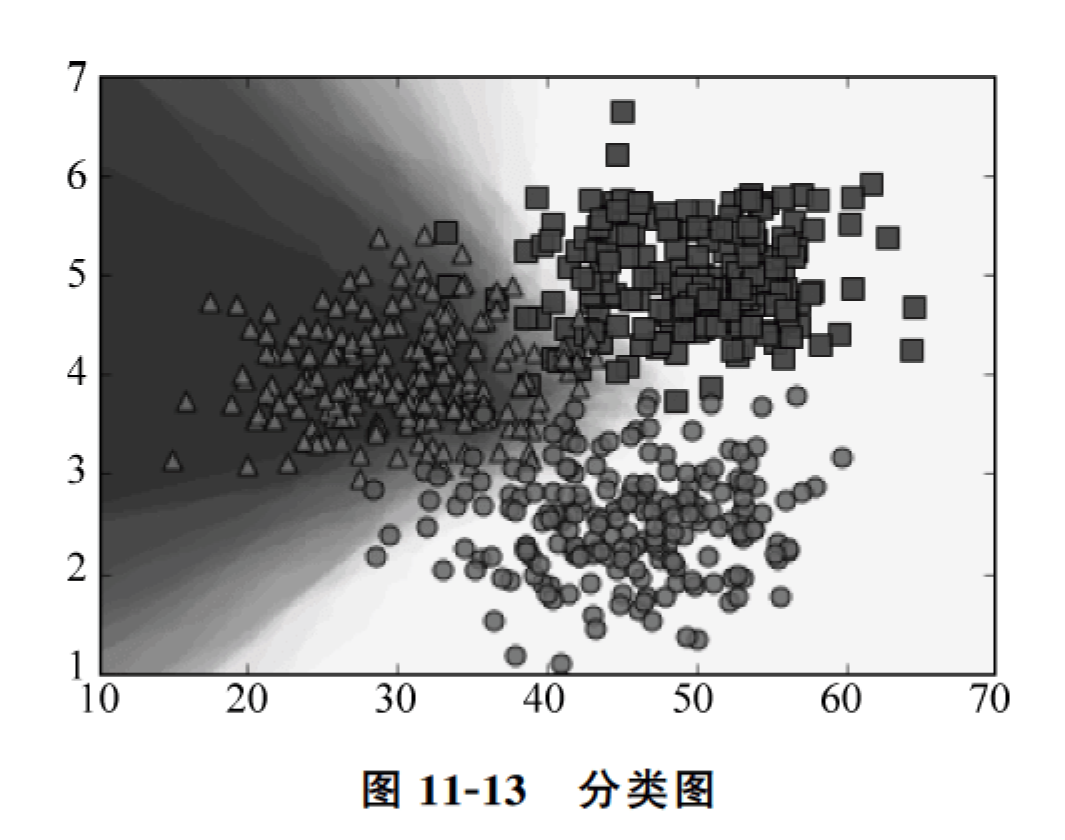


图 11-13 分类图

scikit-learn 程序包的功能非常齐全，使用 KNN 分类进行预测也简单易懂。使用的难点在于将数据整理成函数可以处理的格式的过程偏于烦琐，从输出中读取结论可能也有些麻烦。本节详细介绍了 scikit-learn 程序包中函数的输入、输出以及处理方法，希望读者可以轻松地将这些功能运用到实际应用中。

11.5.2 实际数据的 K 最近邻法分类 Python 应用

1. KNN 分类算法

KNN 分类算法(K-Nearest Neighbors Classification)，又叫 K 最近邻算法，是一个概念极其简单，而分类效果又很优秀的分类算法。其核心思想就是，要确定测试样本属于哪一类，就寻找所有训练样本中与该测试样本“距离”最近的前 K 个样本，然后看这 K 个样本大部分属于哪一类，那么就认为这个测试样本也属于哪一类。简单地说就是让最相似的 K 个样本来投票决定。这里所说的距离，一般最常用的就是多维空间的欧式距离。这里的维度指特征维度，即样本有几个特征就属于几维。

KNN 示意图如图 11-14 所示。

在图 11-14 中，我们要确定测试样本绿色(圆形)属于蓝色(矩形)还是红色(三角形)。显然，当 $K=3$ 时，将以 1∶2 的投票结果分类于红色；而 $K=5$ 时，将以 3∶2 的投票结果分类于蓝色(矩形)。

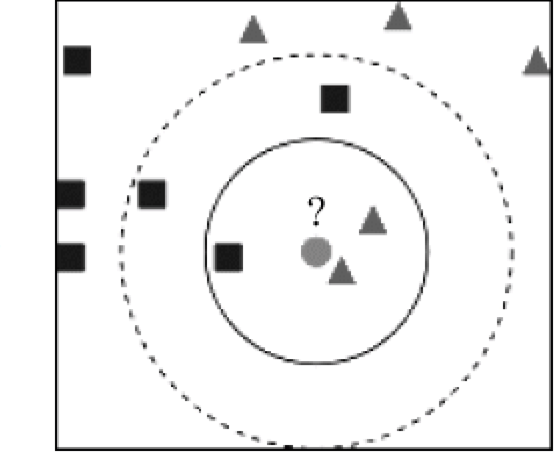


图 11-14 KNN 示意图

KNN 算法简单有效，但没有优化的暴力法效率容易达到瓶颈。如样本个数为 N，特征维度为 D 的时候，该算法时间复杂度呈 $O(DN)$增长。

所以通常 KNN 的实现会把训练数据构建成 KD Tree(K-dimensional tree)，构建过程很快，甚至不用计算 D 维欧氏距离，而搜索速度高达 $O(D*\log(N))$。

不过当 D 维度过高，会产生所谓的“维度灾难”，最终效率会降低到与暴力法一样。因此通常 $D>20$ 以后，最好使用更高效率的 Ball-Tree，其时间复杂度为 $O(D*\log(N))$。人们经过长期的实践发现 KNN 算法虽然简单，但能处理大规模的数据分类，尤其适用于样本分类边界不规则的情况。最重要的是该算法是很多高级机器学习算法的基础。

当然，KNN 算法也存在一些问题。比如如果训练数据大部分都属于某一类，投票算法就有很大问题了。这时候就需要考虑设计每个投票者的权重了。

2. 数据测试文件

先在 G:\\2glkx\\data\\目录下建立数据文件 11-1. txt,测试数据如下:

```
1.5 40.0 thin
1.5 50.0 fat
1.5 60.0 fat
1.6 40.0 thin
1.6 50.0 thin
1.6 60.0 fat
1.6 70.0 fat
1.7 50.0 thin
1.7 60.0 thin
1.7 70.0 fat
1.7 80.0 fat
1.8 60.0 thin
1.8 70.0 thin
1.8 80.0 fat
1.8 90.0 fat
1.9 80.0 thin
1.9 90.0 fat
```

3. Python 代码

scikit-learn 提供了优秀的 KNN 算法支持。使用 Python 代码如下:

```
import NumPy as np
from sklearn import neighbors
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import classification_report
from sklearn.cross_validation import train_test_split
import matplotlib.pyplot as plt
## 数据读入
data = []
labels = []
with open("G:\\2glkx\\data\\al11 - 1.txt") as ifile:
        for line in ifile:
            tokens = line.strip().split(' ')
            data.append([float(tk) for tk in tokens[: - 1]])
            labels.append(tokens[ - 1])
x = np.array(data)
labels = np.array(labels)
y = np.zeros(labels.shape)
## 标签转换为 0/1 '''
y[labels == 'fat'] = 1
## 拆分训练数据与测试数据 '''
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
## ''''' 创建网格以方便绘制
h = .01
x_min, x_max = x[:, 0].min() - 0.1, x[:, 0].max() + 0.1
y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
```

```
                              np.arange(y_min, y_max, h))
##训练KNN分类器 '''
clf = neighbors.KNeighborsClassifier(algorithm = 'kd_tree')
clf.fit(x_train, y_train)
##测试结果的打印'''
answer = clf.predict(x)
print(x)
print(answer)
print(y)
print(np.mean( answer == y))
```

得到如下结果：

```
[[  1.5   40. ]
 [  1.5   50. ]
 [  1.5   60. ]
 [  1.6   40. ]
 [  1.6   50. ]
 [  1.6   60. ]
 [  1.6   70. ]
 [  1.7   50. ]
 [  1.7   60. ]
 [  1.7   70. ]
 [  1.7   80. ]
 [  1.8   60. ]
 [  1.8   70. ]
 [  1.8   80. ]
 [  1.8   90. ]
 [  1.9   80. ]
 [  1.9   90. ]]
[ 1.  1.  1.  0.  0.  1.  1.  0.  0.  1.  1.  0.  0.  1.  1.  1.  1.]
[ 0.  1.  1.  0.  0.  1.  1.  0.  0.  1.  1.  0.  0.  1.  1.  0.  1.]
0.882352941176
```

KNN分类器在众多分类算法中属于最简单的之一，需要注意的地方不多。有这几点需要说明：

(1) KNeighborsClassifier可以设置三种算法：'brute'、'kd_tree'、'ball_tree'。如果不知道用哪个好，设置'auto'让KNeighborsClassifier自己根据输入去决定。

(2) 注意统计准确率时，分类器的score返回的是计算正确的比例，而不是R2。R2一般应用于回归问题。

(3) 本例先根据样本中身高体重的最大最小值，生成了一个密集网格(步长 $h=0.01$)，然后将网格中的每一个点都当成测试样本去测试。

这个数据集达到准确率0.882352941176算是很优秀的结果了。

练　习　题

1. 对本章例题，使用Python重新操作一遍。
2. 对糖尿病数据集，找到最优的正则化参数alpha。(0.016249161908773888)

第12章 时间序列数据分析的 Python 应用

时间序列是时间间隔不变的情况下收集的时间点集合。我们可以分析和了解其长期的发展趋势。但时间序列分析与常见的回归问题有什么不同呢？有两个原因：(1)时间序列是跟时间有关的。所以基于线性回归模型的假设：观察结果是独立的在这种情况下是不成立的。(2)随着上升或者下降的趋势，更多的时间序列出现季节性趋势的形式，如：特定时间框架的具体变化，即：如果我们看到羊毛外衣的销售上升，就一定会在冬季做更多销售。正因为时间序列的固有特性，有各种不同的方法可以对它进行分析。Python 中的 Pandas 和 Statsmodels 有专门处理时间序列对象库，特别是可以存储时间信息和允许人们执行快速合作的 datatime64(ns)类。

12.1 时间序列分析的 ARIMA 建模

常用的时间序列分析模型有 AR 模型、MA 模型、ARMA 模型和 ARIMA 模型等。

12.1.1 时间序列的预处理

得到一个观察值序列后，首先要对它的平稳性和纯随机性进行检验，这两个重要的检验称为序列的预处理。根据检验的结果可以将时间序列分为不同的类型，对不同类型的序列要采用不同的分析方法。

先介绍一下什么是平稳？平稳就是围绕一个常数上下波动且波动范围有限，即：有常数均值和常数方差。如果有明显的趋势或周期性，那它通常不是平稳序列。一个时间序列是否平稳？一般采用三种方法检验：

1. 时间序列图检验

如图 12-1 所示，可见具有很明显的增长趋势，不平稳。

2. 自相关系数和偏相关系数

还以上面的图 12-1 所示的时间序列为例，其自相关和偏相关图，如图 12-2 所示。

从图 12-2 中可见，左边第一个为自相关图(autocorrelation)，第二个为偏相关图(partial correlation)。

平稳的序列的自相关图和偏相关图要么是拖尾，要么是截尾。截尾就是在某阶之后，系数都为 0，怎么理解呢？看图 12-2 中的偏相关图，当阶数为 1 的时候，系数值还是很大的，为

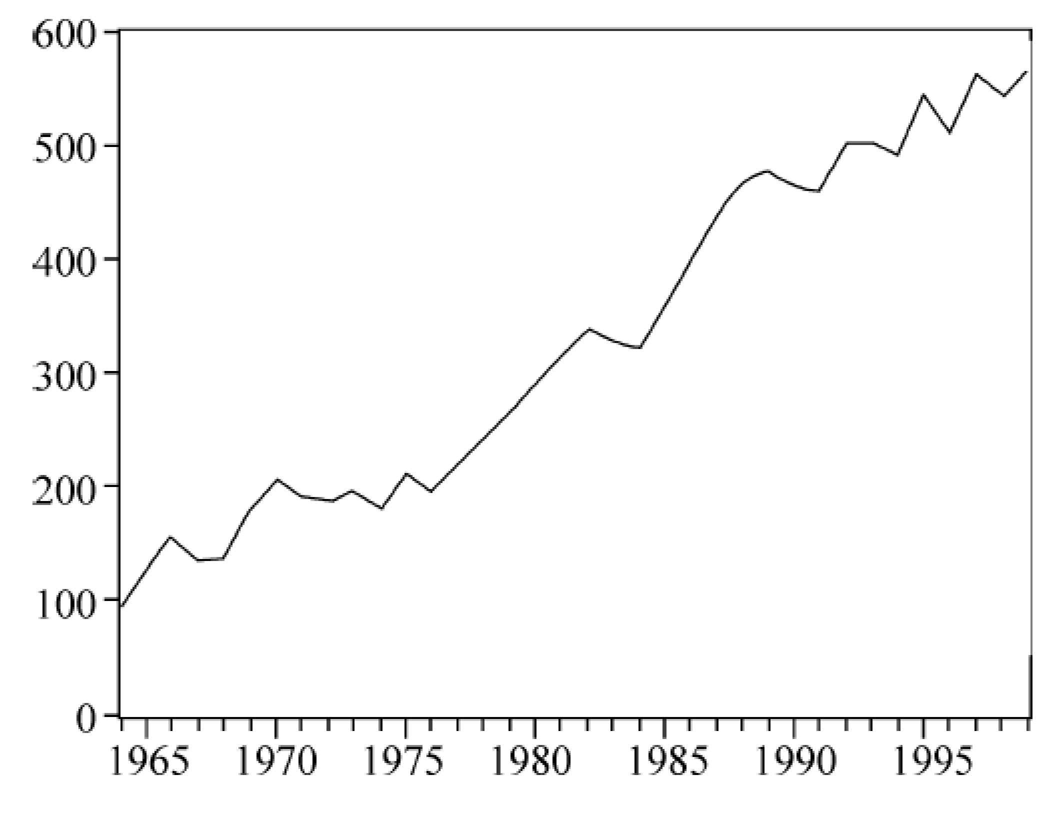


图 12-1　时间序列

Autocorrelation	Partial Correlation		AC	PAC	Q-Stat	Prob
		1	0.914	0.914	32.647	0.000
		2	0.843	0.050	61.269	0.000
		3	0.767	-0.065	85.669	0.000
		4	0.695	-0.021	106.34	0.000
		5	0.609	-0.129	122.69	0.000
		6	0.544	0.065	136.18	0.000
		7	0.476	-0.034	146.88	0.000
		8	0.395	-0.139	154.52	0.000
		9	0.321	-0.019	159.74	0.000
		10	0.242	-0.094	162.82	0.000
		11	0.147	-0.166	163.99	0.000
		12	0.058	-0.038	164.18	0.000
		13	-0.033	-0.116	164.25	0.000
		14	-0.108	0.010	164.98	0.000
		15	-0.168	0.053	166.82	0.000
		16	-0.215	-0.015	169.99	0.000
		17	-0.260	-0.021	174.84	0.000
		18	-0.306	-0.083	181.98	0.000
		19	-0.346	-0.040	191.61	0.000
		20	-0.382	-0.010	204.08	0.000

图 12-2　自相关和偏相关图

0.914；当阶数为 2 的时候突然就变成了 0.050。后面的值都很小，认为是趋于 0，这种状况就是截尾。什么是拖尾？拖尾就是有一个缓慢衰减的趋势，但是不都为 0。

自相关图既不是拖尾也不是截尾。图 12-2 的自相关是一个三角对称的形式，这种趋势是单调趋势的典型图形，说明这个序列不是平稳序列。

3. 单位根检验

单位根检验是指检验序列中是否存在单位根。如果存在单位根就是非平稳时间序列。那么不平稳，怎么办？

答案就是差分，转换为平稳序列。什么是差分？一阶差分指原序列值相距一期的两个序列值之间的减法运算；k 阶差分就是相距 k 期的两个序列值之间相减。如果一个时间序列经过差分运算后具有平稳性，则该序列为差分平稳序列，可以使用 ARIMA 模型进行分析。

还是上面那个序列，两种方法都说明该序列是不平稳的。确定不平稳后，依次进行 1

阶、2 阶、3 阶……差分，直到平稳为止。先来个一阶差分，如图 12-3 所示。

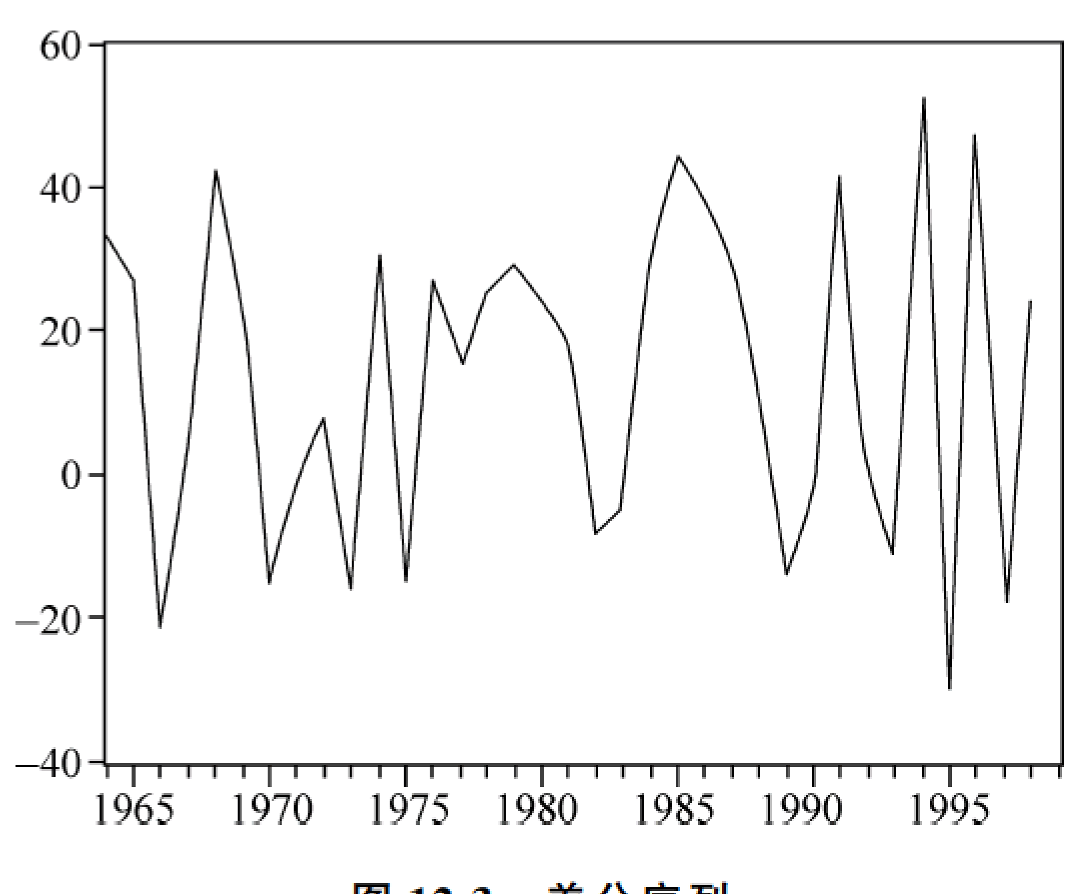


图 12-3　差分序列

从图 12-3 可见，一阶差分的效果不错，看来是平稳的。

平稳性检验过后，下一步是纯随机性检验。

对于纯随机序列，又称白噪声序列，序列的各项数值之间没有任何相关关系，序列在进行完全无序的随机波动，可以终止对该序列的分析。白噪声序列是没有信息可提取的平稳序列。

对于平稳非白噪声序列，它的均值和方差是常数。通常是建立一个线性模型来拟合该序列，借此提取该序列的有用信息。ARMA 模型是最常用的平稳序列拟合模型。

12.1.2　平稳时间序列建模

某个时间序列经过预处理，被判定为平稳非白噪声序列，就可以进行时间序列建模。建模步骤如下。

(1) 计算出该序列的自相关系数(ACF)和偏相关系数(PACF)。

(2) 模型识别，也称模型定阶。根据系数情况从 AR(p)模型、MA(q)模型、ARMA(p,q)模型、ARIMA(p,d,q)模型中选择合适模型，其中 p 为自回归项，d 为差分阶数，q 为移动平均项数。平稳序列的模型选择如表 12-1 所示。

表 12-1　平稳序列的模型选择

自相关系数(ACF)	偏相关系数(PACF)
拖尾	p 阶截尾
q 阶截尾	拖尾
p 阶拖尾	q 阶拖尾

ARIMA 是 ARMA 算法的扩展版，用法类似。

(3) 估计模型中的未知参数的值并对参数进行检验；

(4) 模型检验；

(5) 模型优化；

(6) 模型应用：进行短期预测。

12.2 ARIMA 模型时间序列分析的 Python-Statsmodels 应用

准备基础程序包：Pandas，NumPy，Scipy，Matplotlib，Statsmodels。命令如下：

```
from __future__ import print_function
import Pandas as pd
import NumPy as np
from SciPy import stats
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.graphics.api import qqplot
```

12.2.1 准备数据

```
dta = [10930,10318,10595,10972,7706,6756,9092,10551,9722,10913,11151,8186,6422,6337,
11649,11652,10310,12043,7937,6476,9662,9570,9981,9331,9449,6773,6304,9355,10477,10148,
10395,11261,8713,7299,10424,10795,11069,11602,11427,9095,7707,10767, 12136,12812,12006,
12528,10329,7818,11719,11683,12603,11495,13670,11337,10232, 13261,13230,15535,16837,
19598,14823,11622,19391,18177,19994,14723,15694,13248,9543,12872,13101,15053,12619,
13749,10228,9725,14729,12518,14564,15085,14722,11999,9390,13481,14795,15845,15271,
14686,11054,10395]
dta = np.array(dta,dtype = np.float)          ## 把矩阵变为数组
dta = pd.Series(dta)
dta.index  =  pd.Index(sm.tsa.datetools.dates_from_range('2001','2090'))
dta.plot(figsize = (12,8))
```

得到如图 12-4 所示的图形。

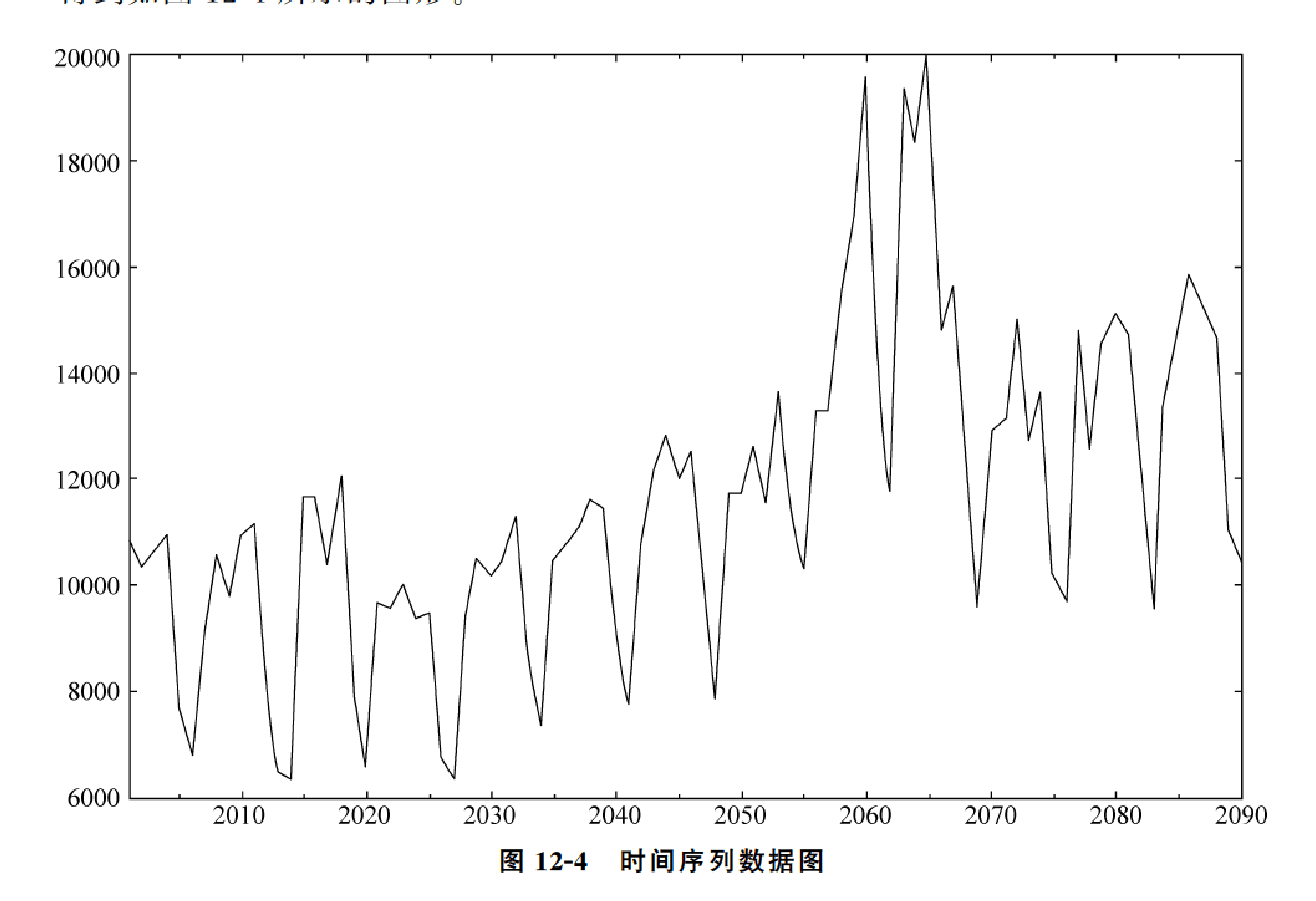


图 12-4 时间序列数据图

12.2.2　差分时间序列

ARIMA 模型对时间序列的要求是平稳型。因此，当我们得到一个非平稳的时间序列时，首先要做的是对时间序列进行差分，直到得到一个平稳时间序列。如果我们对时间序列做 d 次差分才能得到一个平稳序列，那么可以使用 ARIMA(p,d,q)模型，其中 d 是差分次数。下面我们进行一次差分。

```
fig = plt.figure(figsize = (12,8))
ax1 = fig.add_subplot(111)
diff1 = dta.diff(1)
diff1.plot(ax = ax1)
```

得到如图 12-5 所示的图形。

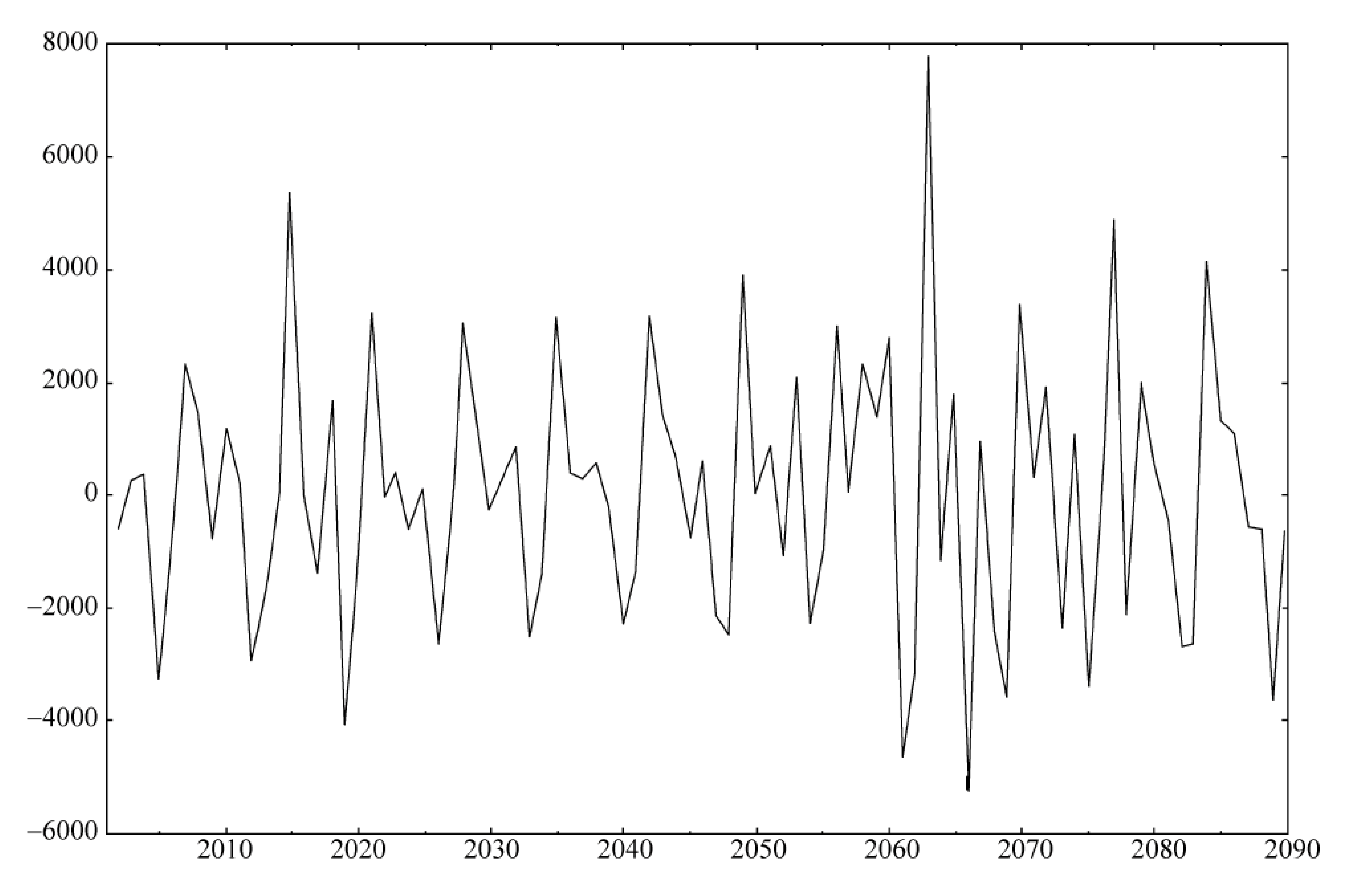


图 12-5　一次差分后的时间序列数据图

进行二次差分：

```
fig = plt.figure(figsize = (12,8))
ax1 = fig.add_subplot(111)
diff2 = dta.diff(2)
diff2.plot(ax = ax1)
```

得到如图 12-6 所示的图形。

从图 12-5、12-6 中可以看到，二次差分的序列效果与一次差分后序列差不多，因此用一次差分后的序列。

12.2.3　选择合适的 p,q

现在我们已经得到了一个平稳的时间序列(一次差分后的数据序列)，接来下就是选择

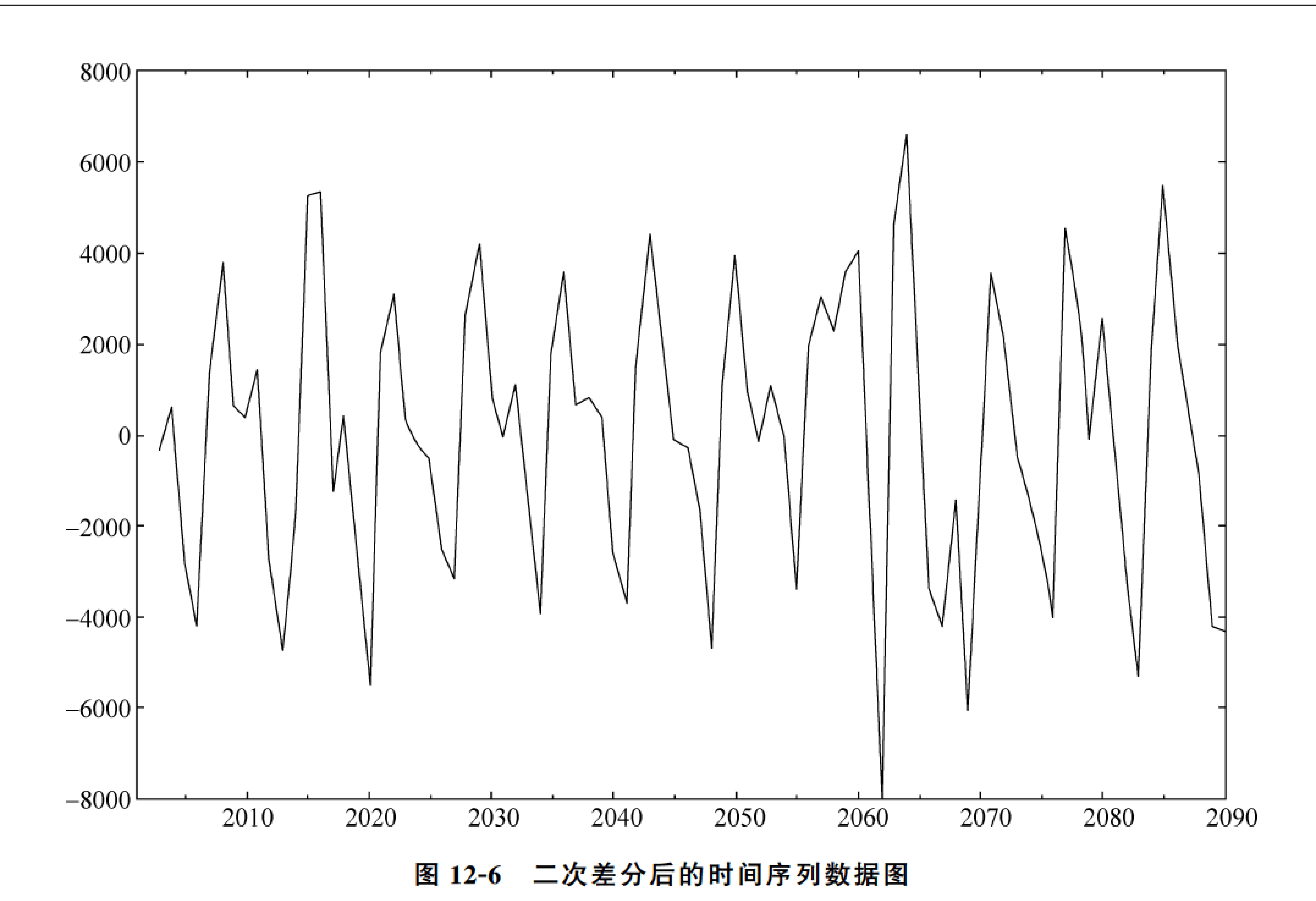


图 12-6 二次差分后的时间序列数据图

合适的 ARIMA 模型,即 ARIMA 模型中合适的 p,q。

1. 第一步,我们要先检查平稳时间序列的自相关图和偏自相关图

```
diff1 = dta.diff(1)#使用一阶差分的时间序列数据
fig = plt.figure(figsize=(12,8))
ax1=fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(dta,lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(dta,lags=40,ax=ax2)
```

其中 lags 表示滞后的阶数,以上分别得到如图 12-7 所示的图形。

通过观察图 12-6 和图 12-7,可以得到:

(1) 自相关图显示滞后有三个阶超出了置信边界;

(2) 偏相关图显示在滞后 1 至 7 阶(lags 1,2,…,7)时的偏自相关系数超出了置信边界,从 lag 7 之后偏自相关系数值缩小至 0。

2. 模型选择

根据图 12-6 和图 12-7,猜测有以下模型可供选择。

(1) ARMA(0,1)模型:即自相关图在滞后 1 阶之后缩小为 0,且偏自相关缩小至 0,则是一个阶数 q=1 的移动平均模型;

(2) ARMA(7,0)模型:即偏自相关图在滞后 7 阶之后缩小为 0,且自相关缩小至 0,则是一个阶层 p=7 的自回归模型;

(3) ARMA(7,1)模型:即使得自相关和偏自相关都缩小至零。则是一个混合模型。

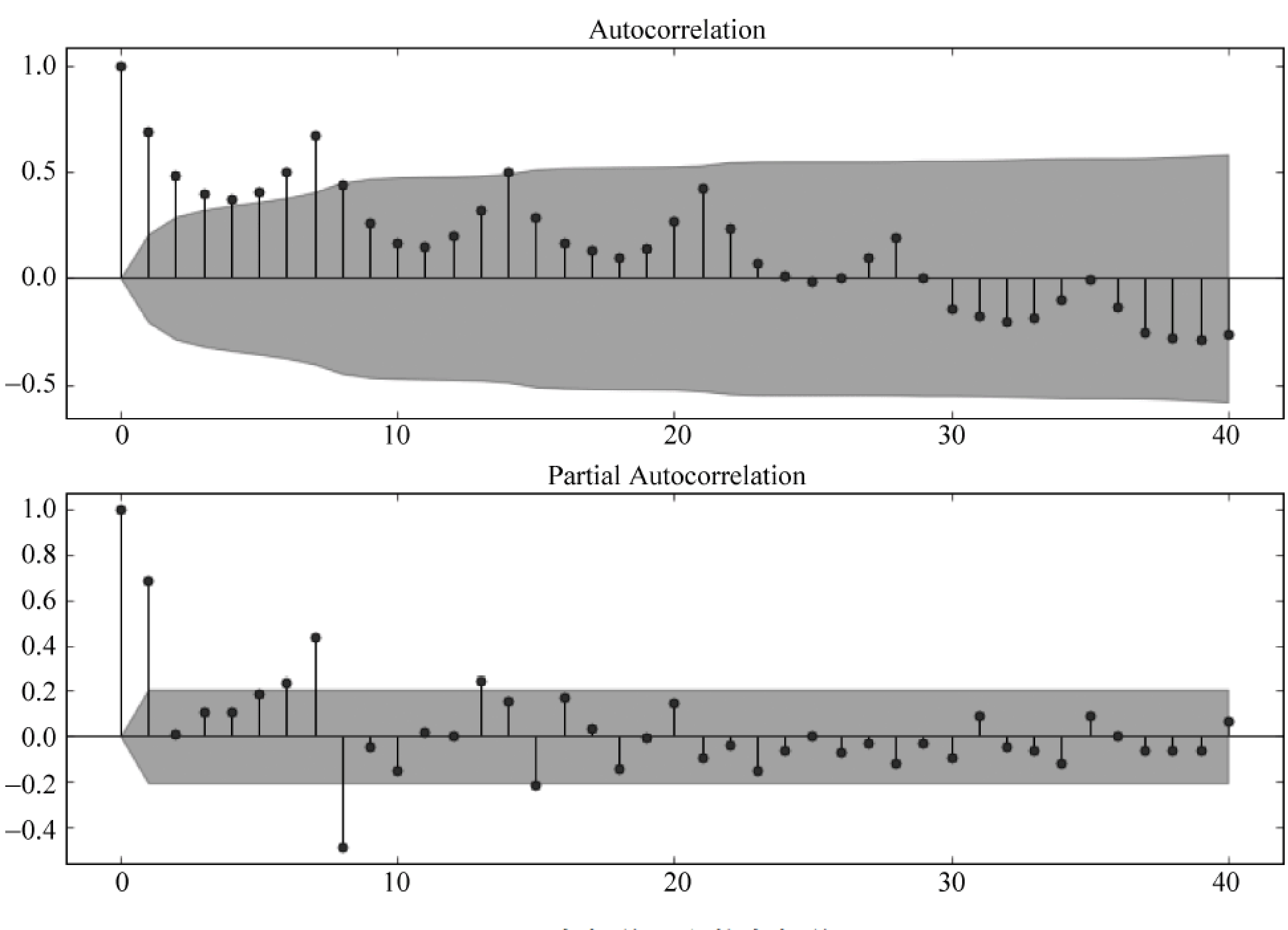


图 12-7　自相关图和偏自相关图

……

(4) 还可以有其他供选择的模型(实际上选了 ARMA(8,0))。

现在有以上这么多可供选择的模型，我们通常采用 ARMA 模型的赤池信息准则(AIC)。我们知道：增加自由参数的数目提高了拟合的优良性，AIC 鼓励数据拟合的优良性但是尽量避免出现过度拟合(overfitting)的情况。所以优先考虑的模型应是 AIC 值最小的那一个。赤池信息准则的方法是寻找可以最好地解释数据但包含最少自由参数的模型。不仅包括 AIC 准则，目前选择模型通常有如下准则：

① AIC=−2 ln(L)+2 k 中文名字：赤池信息量 akaike information criterion；

② BIC=−2 ln(L)+ln(n) * k 中文名字：贝叶斯信息量 bayesian information criterion；

③ HQ=−2 ln(L) + ln(ln(n)) * k hannan-quinn criterion。

构造这些统计量所遵循的统计思想是一致的，就是在考虑拟合残差的同时，根据自变量个数加以“惩罚”。但要注意的是，这些准则不能说明某一个模型的精确度，也就是说，对于三个模型 A、B、C，我们能够判断出 C 模型是最好的，但不能保证 C 模型能够很好地刻画数据，因为有可能三个模型都是糟糕的。

```
arma_mod70 = sm.tsa.ARMA(dta,(7,0)).fit()
print(arma_mod70.aic,arma_mod70.bic,arma_mod70.hqic)
arma_mod01 = sm.tsa.ARMA(dta,(0,1)).fit()
print(arma_mod10.aic,arma_mod30.bic,arma_mod30.hqic)
arma_mod71 = sm.tsa.ARMA(dta,(7,1)).fit()
print(arma_mod71.aic,arma_mod71.bic,arma_mod71.hqic)
arma_mod80 = sm.tsa.ARMA(dta,(8,0)).fit()
print(arma_mod80.aic,arma_mod80.bic,arma_mod80.hqic)
```

```
1619.19181153 1641.69009856 1628.26444334
1657.21726267 1664.71669168 1660.24147327
1605.68656481 1630.68466152 1615.76726682
1597.93600985 1622.93410655 1608.01671185
```

这样的话应该是 ARMA(8,0)模型拟合效果最好。

3. 检验残差序列

```
resid = arma_mod80.resid
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(resid.values.squeeze(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(resid, lags=40, ax=ax2)
```

得到如图 12-8 所示的图形。

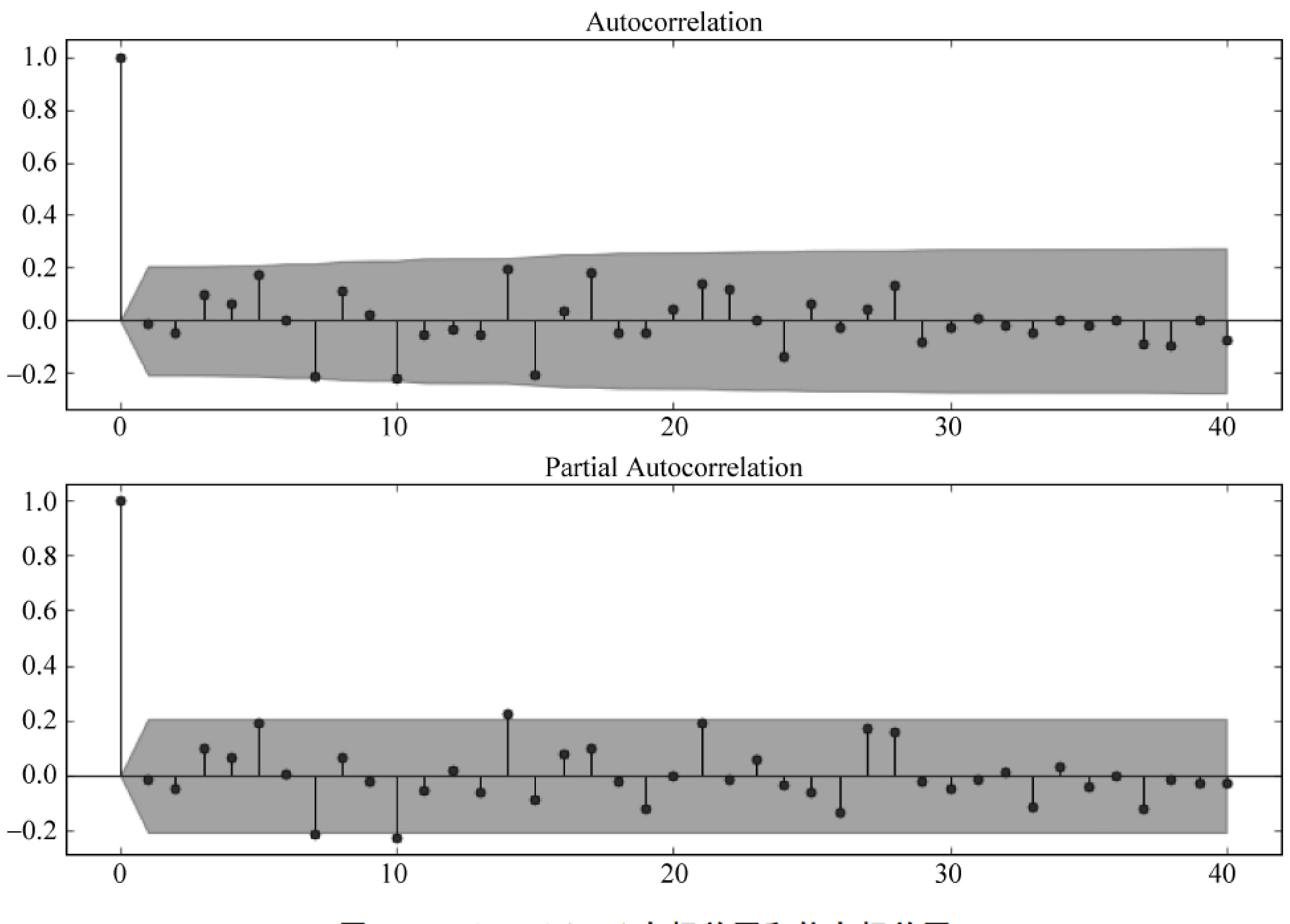
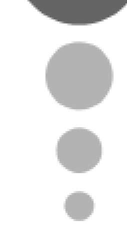


图 12-8 ARMA(8,0)自相关图和偏自相关图

从图 12-8 中,可以看到序列残差基本为白噪声

进一步进行德宾-沃森(Durbin-Watson)检验。德宾-沃森检验,简称 DW 检验,是目前检验自相关性最常用的方法,但它只适用于检验一阶自相关性。因为自相关系数 ρ 的值介于-1 和 1 之间,所以 $0\leqslant DW\leqslant 4$。并且 $DW=O\times\rho=1$ 即存在正自相关性;$DW=4\times\rho=-1$ 即存在负自相关性;$DW=2\times\rho=0$ 即不存在(一阶)自相关性。因此,当 DW 值显著地接近于 0 或 4 时,则存在自相关性;而接近于 2 时,则不存在(一阶)自相关性。这样只要知道 DW 统计量的概率分布,在给定的显著水平下,根据临界值的位置就可以对原假设 H_0 进行检验。




```
print(sm.stats.durbin_watson(arma_mod80.resid.values))
2.02277213502
```

结果=2.02277213502,所以残差序列不存在自相关性。

4. 观察是否符合正态分布

这里使用 QQ 图,它用于直观验证一组数据是否来自某个分布,或者验证某两组数据是否来自同一(族)分布。

```
print(stats.normaltest(resid))
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111)
fig = qqplot(resid, line='q', ax=ax, fit=True)
```

得到如图 12-9 所示的图形。

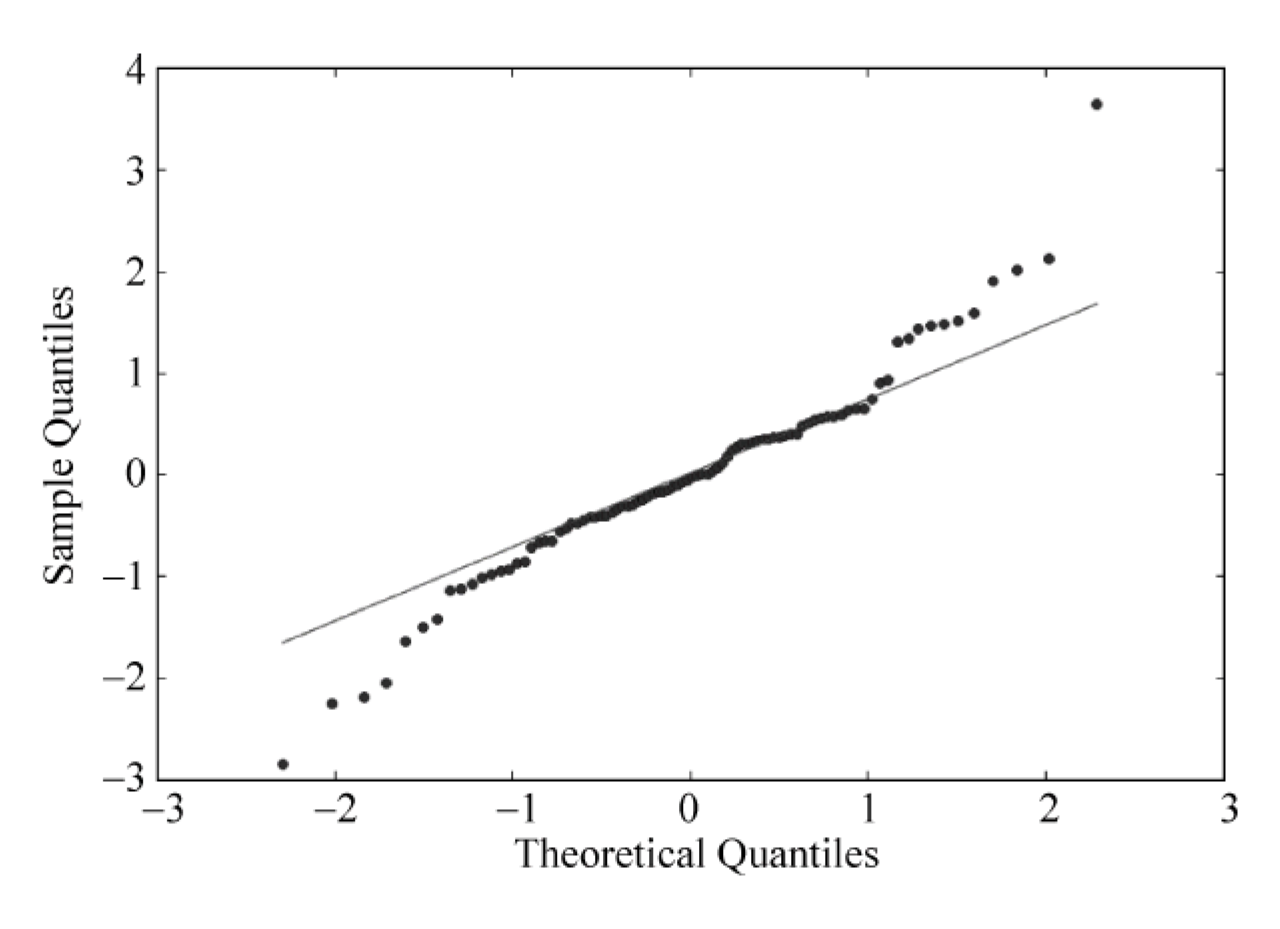


图 12-9　正态分布检验

从图 12-9 可见,基本符合正态分布。

5. 残差序列 Ljung-Box 检验(Q 检验)

```
r,q,p = sm.tsa.acf(resid.values.squeeze(), qstat=True)
data = np.c_[range(1,41), r[1:], q, p]
table = pd.DataFrame(data, columns=['lag', "AC", "Q", "Prob(>Q)"])
print(table.set_index('lag'))
```

得到如下结果:

```
            AC         Q   Prob(>Q)
lag
1.0  -0.014468  0.019475  0.889013
2.0  -0.045573  0.214888  0.898127
3.0   0.101607  1.197454  0.753615
4.0   0.063476  1.585377  0.811418
5.0   0.176140  4.607596  0.465618
6.0   0.005123  4.610182  0.594689
```

```
7.0   -0.208965    8.966309  0.255084
8.0    0.115859   10.321747  0.243166
9.0    0.020846   10.366170  0.321663
10.0  -0.220057   15.378186  0.118870
11.0  -0.050613   15.646675  0.154763
12.0  -0.031538   15.752259  0.202848
13.0  -0.055134   16.079128  0.244883
14.0   0.195246   20.232322  0.122985
15.0  -0.204286   24.839599  0.052139
16.0   0.034589   24.973464  0.070295
17.0   0.181380   28.704982  0.037345
18.0  -0.043261   28.920203  0.049363
19.0  -0.045300   29.159517  0.063517
20.0   0.044260   29.391235  0.080337
21.0   0.142092   31.814067  0.061136
22.0   0.118281   33.517592  0.054955
23.0   0.004485   33.520077  0.072423
24.0  -0.133530   35.756960  0.057939
25.0   0.061820   36.243786  0.067981
26.0  -0.021852   36.305567  0.086191
27.0   0.047349   36.600216  0.102775
28.0   0.131287   38.902103  0.082510
29.0  -0.080830   39.788943  0.087447
30.0  -0.026237   39.883938  0.107132
31.0   0.011392   39.902152  0.131262
32.0  -0.015232   39.935275  0.158191
33.0  -0.042672   40.199784  0.181490
34.0   0.006094   40.205275  0.214516
35.0  -0.015923   40.243444  0.249286
36.0   0.000432   40.243473  0.287867
37.0  -0.083854   41.341975  0.286677
38.0  -0.091988   42.689361  0.276570
39.0   0.002412   42.690305  0.315490
40.0  -0.071636   43.540115  0.323196
```

从上可见，prob 值均大于 0.05，所以残差序列不存在自相关性。

12.2.4 预测

```
predict_dta = arma_mod80.predict('2090', '2100', dynamic=True)
print(predict_dta)
fig, ax = plt.subplots(figsize=(12, 8))
ax = dta.ix['2000':].plot(ax=ax)
fig = arma_mod80.plot_predict('2090', '2100', dynamic=True, ax=ax, plot_insample=False)
```

得到如下结果：

```
2090-12-31    9543.342785
2091-12-31    12908.557524
2092-12-31    13981.006838
```

```
2093 - 12 - 31     14501.081732
2094 - 12 - 31     13893.733284
2095 - 12 - 31     13249.430373
2096 - 12 - 31     10961.736141
2097 - 12 - 31     10074.458409
2098 - 12 - 31     12685.025109
2099 - 12 - 31     13476.428895
2100 - 12 - 31     13615.435265
Freq: A - DEC, dtype: float64
```

得到如图 12-10 所示的图形。

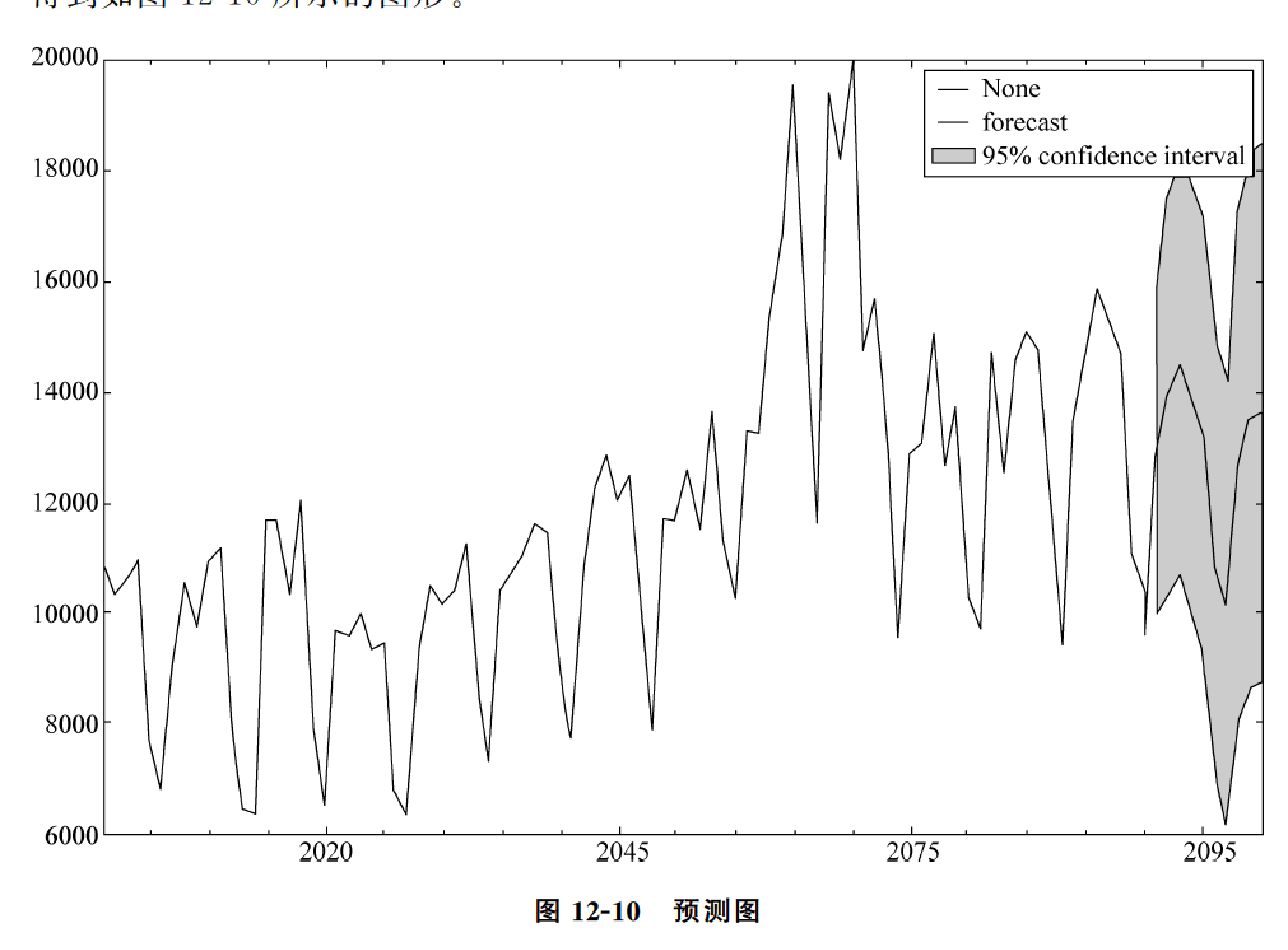


图 12-10　预测图

12.3　时间序列数据分析 ARIMA 模型的 Python 应用

下面以某店铺 2016/1/1～2016/2/5 的销售数据为例，以此来建模预测 2016/2/6～2016/2/10 的销售数据，数据如表 12-2 所示。

表 12-2　销 售 数 据

日期	销量	日期	销量
2016/1/1	3023	2016/1/19	3421
2016/1/2	3039	2016/1/20	3443
2016/1/3	3056	2016/1/21	3428
2016/1/4	3138	2016/1/22	3554

续表

日期	销量	日期	销量
2016/1/5	3188	2016/1/23	3555
2016/1/6	3224	2016/1/24	3556
2016/1/7	3226	2016/1/25	3557
2016/1/8	3029	2016/1/26	3553
2016/1/9	2859	2016/1/27	3559
2016/1/10	2870	2016/1/28	3630
2016/1/11	2910	2016/1/29	3700
2016/1/12	3012	2016/1/30	3800
2016/1/13	3142	2016/1/31	3900
2016/1/14	3252	2016/2/1	4000
2016/1/15	3342	2016/2/2	4200
2016/1/16	3365	2016/2/3	4400
2016/1/17	3339	2016/2/4	4600
2016/1/18	3345	2016/2/5	4800

```
#arima 时间序列分析模型
import Pandas as pd
#参数初始化
discfile = 'G:/2glkx/data/al12-2.xls'
forecastnum = 5
#读取数据,指定日期列为指标,Pandas 自动将"日期"列识别为 Datetime 格式
data = pd.read_excel(discfile, index_col = u'date')
#时序图
import matplotlib.pyplot as plt
#用来正常显示中文标签
plt.rcParams['font.sans-serif'] = ['SimHei']
#用来正常显示负号
plt.rcParams['axes.unicode_minus'] = False
data.plot()
plt.show()
```

得到如图 12-11 所示的图形。

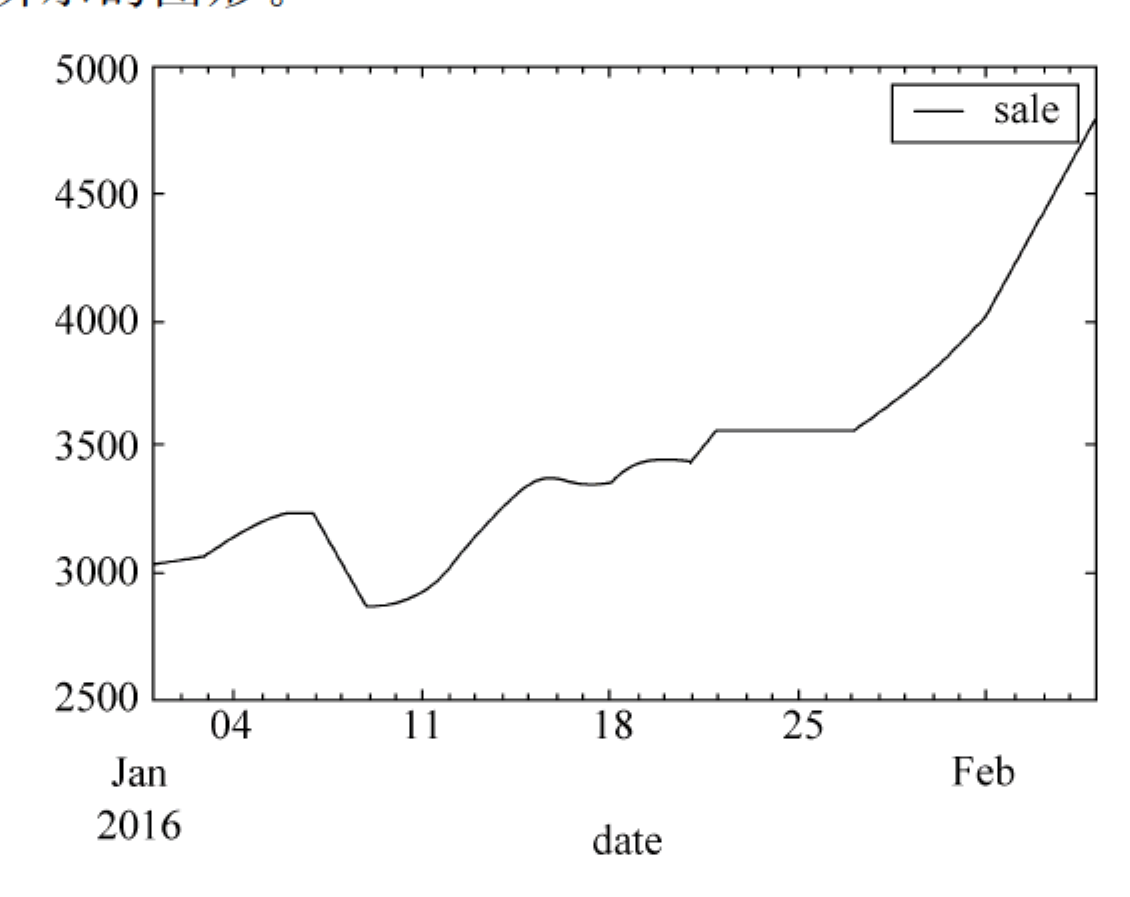
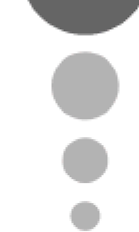


图 12-11 销量数据




```
#自相关图
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(data).show()
```

得到如图 12-12 所示的图形。

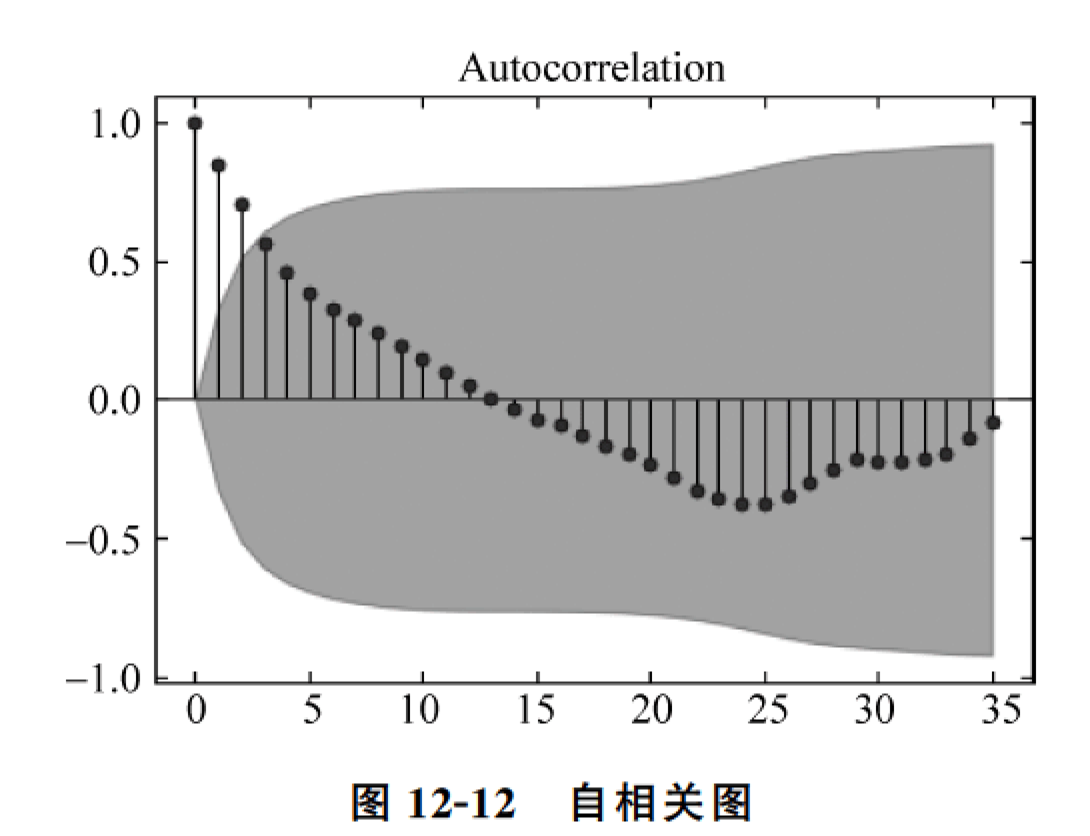


图 12-12　自相关图

```
#平稳性检测
from statsmodels.tsa.stattools import adfuller as ADF
print(u'result:', ADF(data[u'sale']))

#返回值依次为 adf、pvalue、usedlag、nobs、critical values、icbest、regresults、resstore
(u'result:', (1.0533790403327086, 0.99479841693347348, 1, 34, {'5%': -2.9512301791166293,
'1%': -3.639224104416853, '10%': -2.6144469896193772}, 272.13072719088905))
```

整理上述结果，如表 12-3 所示。

表 12-3　ADF 结果

原始序列的单位根(adf)检验				
adf	cValue			p 值
	1%	5%	10%	
1.053379	−3.6392	−2.9512	−2.6144	0.99480

从表 12-3 中可见，p 值大于 3 个关键值，p 值显著大于 0.05，该序列为非平稳序列。

```
#差分后的结果
D_data = data.diff().dropna()
D_data.columns = [u'chafen']
#时序图
D_data.plot()
plt.show()
```

得到如图 12-13 所示的图形。

```
#自相关图
plot_acf(D_data).show()
plt.show()
```

得到如图 12-14 所示的自相关图形。

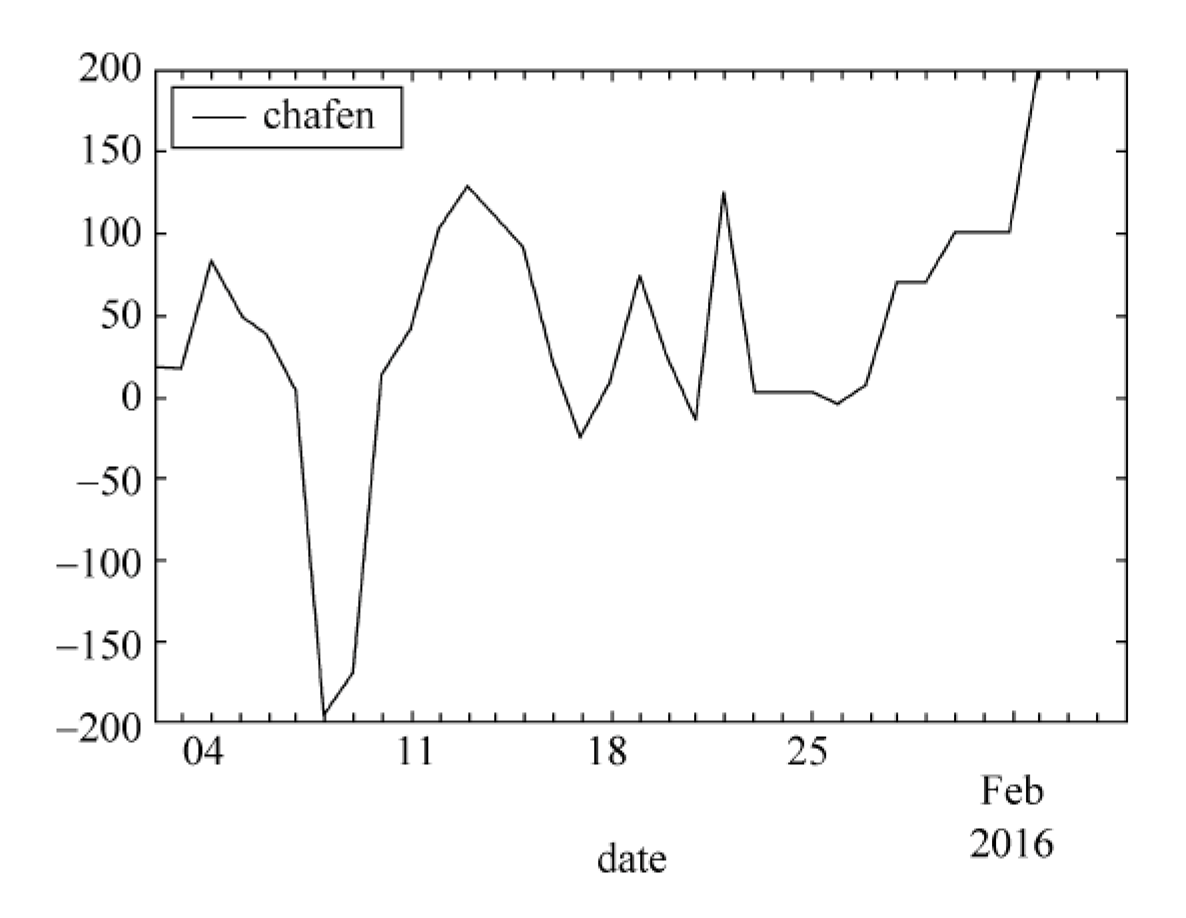


图 12-13 差分后序列

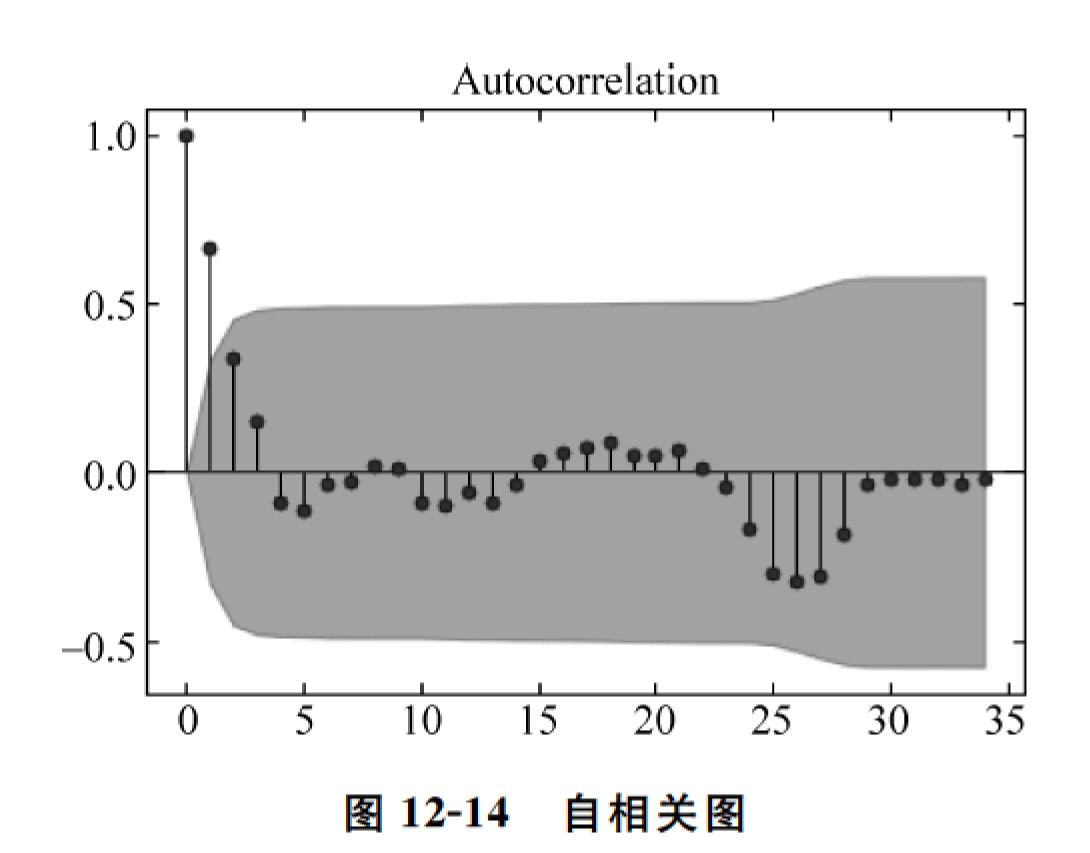


图 12-14 自相关图

```
from statsmodels.graphics.tsaplots import plot_pacf
#偏自相关图
plot_pacf(D_data).show()
```

得到如图 12-15 所示的偏相关图形。

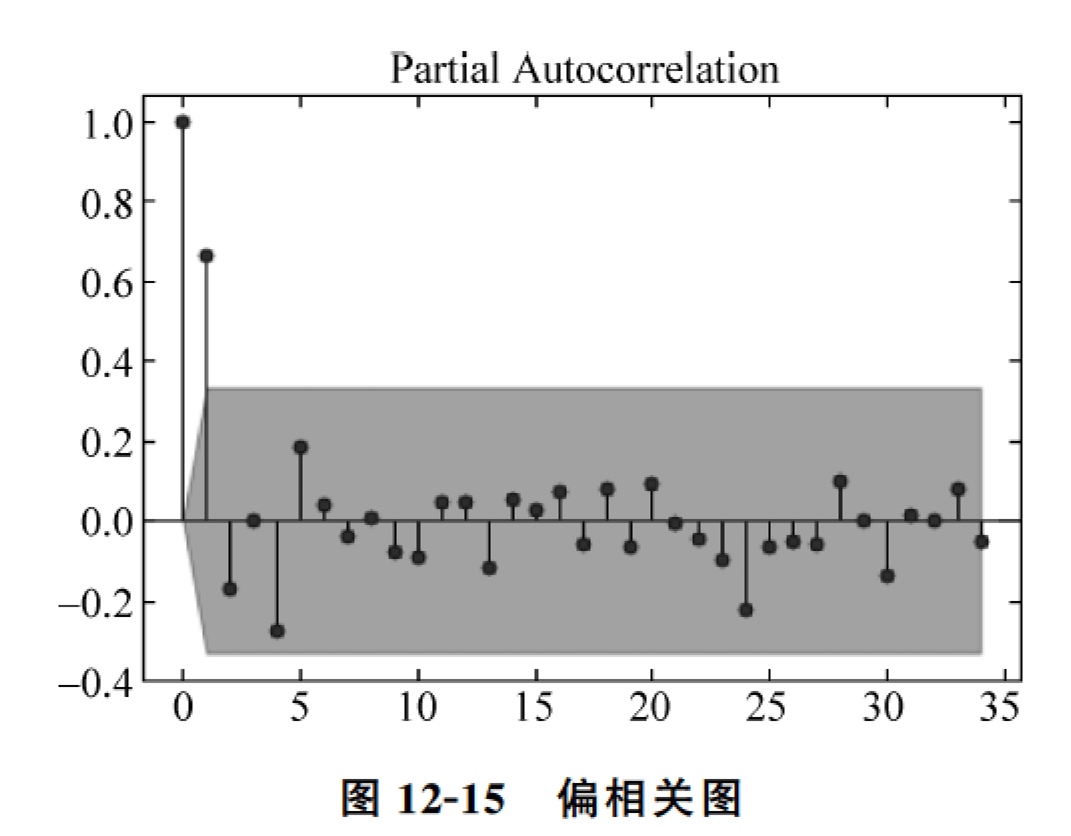
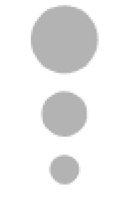


图 12-15 偏相关图

```
#ADF 平稳性检测
print(u'result:', ADF(D_data[u'chafen']))
```




```
(u'result:', (-2.0689698606420945, 0.25717359151753916, 0, 34, {'5%': -2.9512301791166293, '
1%': -3.639224104416853, '10%': -2.6144469896193772}, 260.91255011693556))
```

整理结果如表 12-4 所示。

表 12-4　ADF 结 果

原始序列的单位根(adf)检验				
adf	cValue			p 值
	1%	5%	10%	
−2.0690	−3.6392	−2.9512	−2.6145	0.2572

p 值显著大于 0.05,一阶差分序列为非平稳序列。

```
#差分后的结果
DD_data = D_data.diff().dropna()
DD_data.columns = [u'cf']
print(u'result:', ADF(DD_data[u'cf']))
(u'result:', (-5.7919083675757923, 4.8502514924723576e-07, 0, 33, {'5%': -2.954126991123355,
'1%': -3.6461350877925254, '10%': -2.6159676124885216}, 251.34579174861824))
```

整理结果如表 12-5 所示。

表 12-5　ADF 结 果

原始序列的单位根(adf)检验				
adf	cValue			p 值
	1%	5%	10%	
−5.7919	−3.6461	−2.9541	−2.6160	0.0000

p 值显著小于 0.05,二阶差分序列为平稳序列。

```
#白噪声检验
from statsmodels.stats.diagnostic import acorr_ljungbox
#返回统计量和 p 值
print(u'result:', acorr_ljungbox(DD_data, lags=1))
(u'result:', (array([ 0.05777836]), array([ 0.81004242])))
```

二阶差分后序列的白噪声检验

```
stat     p 值
0.81     0.057
```

p 值小于 0.10,所以二阶差分后的序列为平稳非白噪声序列。

```
from statsmodels.tsa.arima_model import ARIMA
#定阶
#一般阶数不超过 length/10
pmax = int(len(DD_data)/10)
#一般阶数不超过 length/10
qmax = int(len(D_data)/10)
#bic 矩阵
bic_matrix = []
```

```
for p in range(pmax + 1):
    tmp = []
    for q in range(qmax + 1):
#存在部分报错,所以用 try 来跳过报错。
      try:
          tmp.append(ARIMA(D_data, (p,1,q)).fit().bic)
      except:
          tmp.append(None)
    bic_matrix.append(tmp)
#从中可以找出最小值
bic_matrix = pd.DataFrame(bic_matrix)
#先用 stack 展平,然后用 idxmin 找出最小值位置。
p,q = bic_matrix.stack().idxmin()
print(u'BIC min p & q: %s、%s' % (p,q))
```

取 BIC 信息量达到最小的模型阶数,结果 p 为 0,q 为 1,定阶完成。

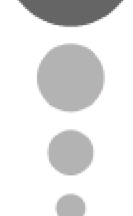
```
#建立 ARIMA(0, 1, 1)模型
model = ARIMA(D_data, (0,1,1)).fit()
#给出一份模型报告
model.summary2()
```

得到如下结果:

```
                          Results: ARIMA
=====================================================================
Model:               ARIMA              Log-Likelihood:     -191.18
Dependent Variable: D.cf               Scale:              1.0000
Date:               2016-10-01 14:59   Method:             css-mle
No. Observations:   34                 Sample:             01-03-2016
Df Model:           2                                      02-05-2016
Df Residuals:       32                 S.D. of innovations: 66.953
AIC:                388.3632           HQIC:               389.925
BIC:                392.9423
-------------------------------------------------------------------
                 Coef.    Std.Err.      t      P>|t|    [0.025    0.975]
-------------------------------------------------------------------
const           5.4243   10.7605     0.5041   0.6176  -15.6659   26.5145
ma.L1.D.cf     -0.0648    0.2182    -0.2969   0.7684   -0.4924    0.3628
---------------------------------------------------------------------
                 Real        Imaginary         Modulus        Frequency
---------------------------------------------------------------------
MA.1             15.4376     0.0000            15.4376          0.0000
=====================================================================
#作为期 5 天的预测,返回预测结果、标准误差、置信区间。
model.forecast(5)
```

最终模型 D_data 预测值如下:

```
(array([ 205.7982486 , 211.22255025, 216.64685189, 222.07115354,
         227.49545519]),
array([   66.95282074,   91.67016965,   111.01423242,   127.45548085,
```




```
          142.00577241]),
array([[  74.57313128, 337.02336592],
       [  31.55231928, 390.89278121],
       [  -0.93704543, 434.23074922],
       [ -27.73699856, 471.87930564],
       [ -50.83074433, 505.8216547 ]]))
```

因此，原时间序列的预测值如下：

2016/2/6	2016/2/7	2016/2/8	2016/2/9	2016/2/9
5006	5217	5434	5656	5883

练　习　题

对本章的数据，使用 Python 重新操作一遍。

第13章 量化金融数据分析的 Python 应用

本章将介绍 7 个量化金融的实例，它们包括：(1)战胜股票市场策略可视化的 Python 应用；(2)股票数据正态性检验的 Python 应用；(3)标准均值方差模型及其 Python 应用；(4)投资组合有效边界的 Python 绘制；(5)Markowitz 投资组合优化的 Python 应用；(6)蒙特卡罗模拟股票期权定价的 Python 应用；(7)蒙特卡罗模拟期权价格稳定性的 Python 应用。

13.1 战胜股票市场策略可视化的 Python 应用

13.1.1 使用 Pandas 导入数据

```
import Pandas.io.data as web
from Pandas import DataFrame
data_feed = {}
data_feed[1] = web.get_data_yahoo('AAPL', '05/1/2016', '10/1/2016')
data_feed[2] = web.get_data_yahoo('FB', '05/1/2016', '10/1/2016')
data_feed[3] = web.get_data_yahoo('GOOG', '05/1/2016', '10/1/2016')
data_feed[4] = web.get_data_yahoo('SPLK', '05/1/2016', '10/1/2016')
data_feed[5] = web.get_data_yahoo('YELP', '05/1/2016', '10/1/2016')
data_feed[6] = web.get_data_yahoo('GG', '05/1/2016', '10/1/2016')
data_feed[7] = web.get_data_yahoo('BP', '05/1/2016', '10/1/2016')
data_feed[8] = web.get_data_yahoo('SCPJ', '05/1/2016', '10/1/2016')
data_feed[9] = web.get_data_yahoo('JNJ', '05/1/2016', '10/1/2016')
price = DataFrame({tic: data['Adj Close'] for tic, data in data_feed.iteritems()})
volume = DataFrame({tic: data['Volume'] for tic, data in data_feed.iteritems()})
```

13.1.2 收益率

要确定收益率百分比并进行分析，可以调用 return DataFrame 方法和 Plot 方法。这可以通过调用 sum 对 DataFrame 中的各列求和来实现，该函数执行了大量工作来创建图 13-1 所示的图表。

```
returns = price.pct_change()
import matplotlib.pyplot as plt
returns.sum().plot(kind='bar',title="% return for Year")
plt.show()
```

最后得到如图 13-1 所示的图形。

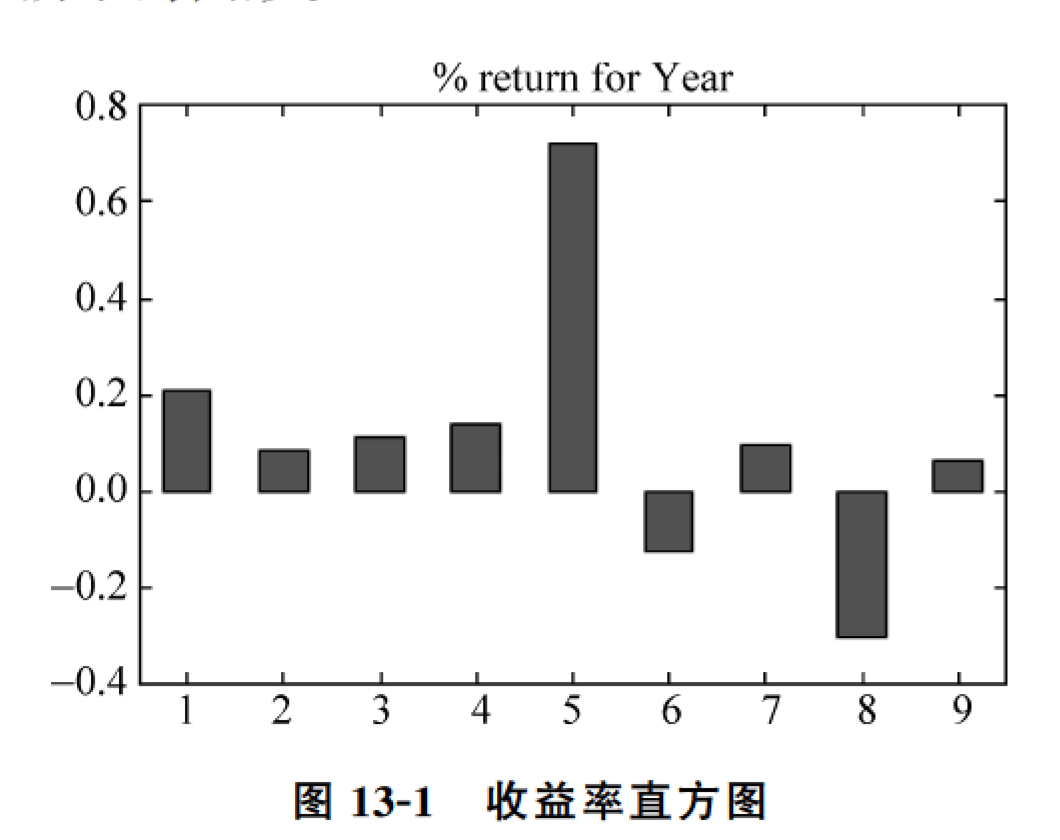


图 13-1　收益率直方图

如图 13-1 所示，SCPJ 进行了 IPO，并且年初至今它的损失接近 IPO 值的 40%。相比之下，YELP（在同一个行业中）获利几乎为 80%。事后看来，卖空 SCPJ 而买进 Yelp 几乎可以让原始投资翻倍。

13.1.3　原始输出总和

sum()命令的文本输出在该代码中展示了年收益的实际原始值：

```
In [12]: returns.sum()
Out[15]:
1      0.209309
2      0.084193
3      0.112291
4      0.137510
5      0.721123
6     -0.124810
7      0.095181
8     -0.302034
9      0.062589
dtype: float64
```

13.1.4　创建一幅日收益率柱状图

考虑数据的另一个方法是创建全年日收益率变化的柱状图，了解这是否反映了数据的底层洞察。这非常简单，代码如下：

```
returns.diff().hist()
plt.show()
```

得到输出结果如下，图形如图 13-2 所示。

```
Out[16]:
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0B7FE1D0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0B9697B0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0BB18210>],
```

```
[<matplotlib.axes._subplots.AxesSubplot object at 0x0BB4D1F0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x0BB89B70>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x0BC3C750>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x0BC8C0F0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x0BCD5770>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x0BD8FD50>]], dtype=object)
```

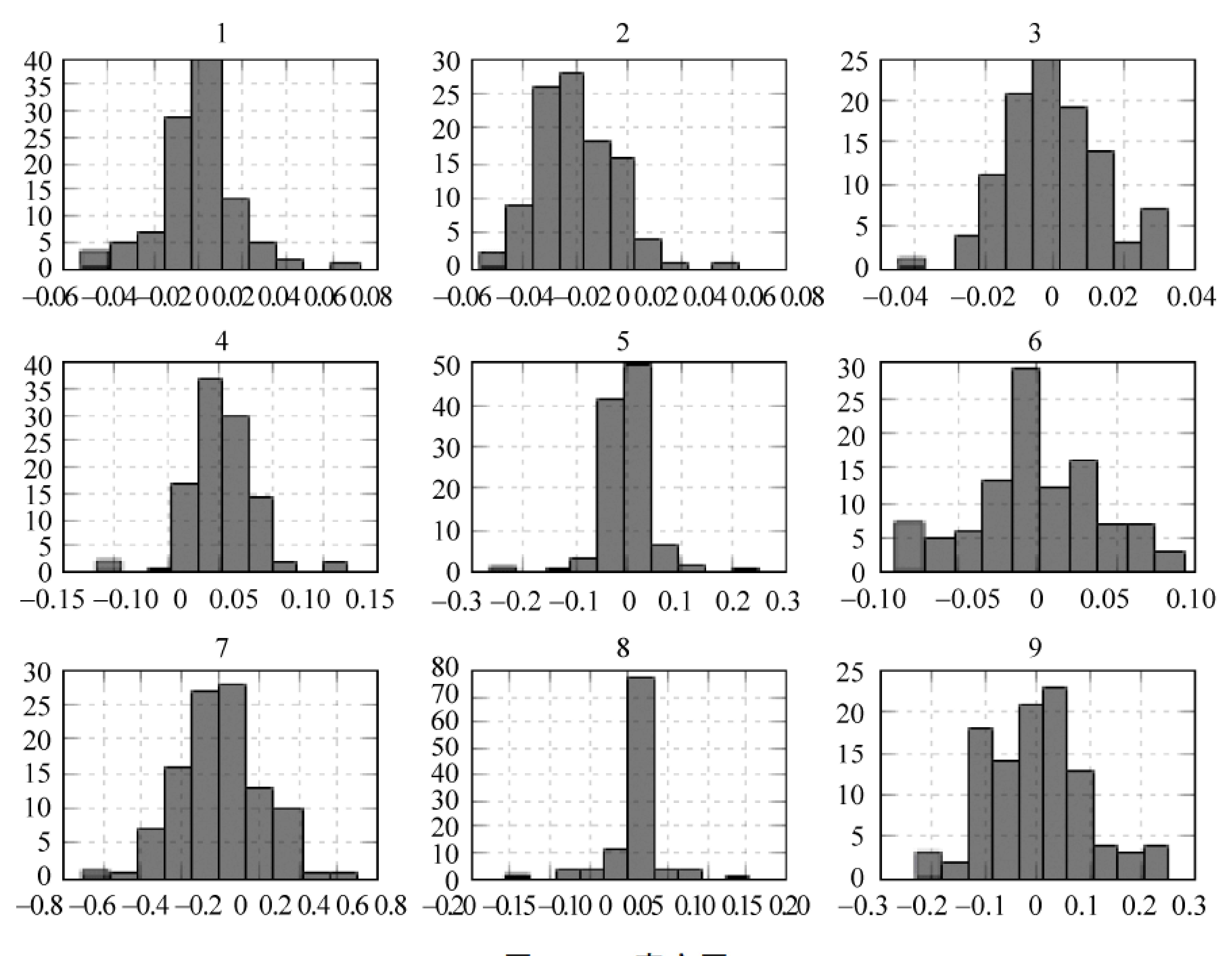


图 13-2 直方图

13.1.5 Pandas 投资组合相关性的年度线性图

另一个查看数据的方法是记下日收益率并绘制年度线性图。下面的代码展示了如何操作：

```
returns.plot(title="% Daily Change for Year")
plt.show()
```

得到如图 13-3 所示的图形。

13.1.6 各资产收益率的累积和

这种简单图表的问题是不太容易理解图中的信息。处理时间序列数据的方法是使用 cumsum 函数，将数据绘成图表：

```
ts = returns.cumsum()
plt.figure(); ts.plot(); plt.legend(loc='upper left')
plt.show()
```

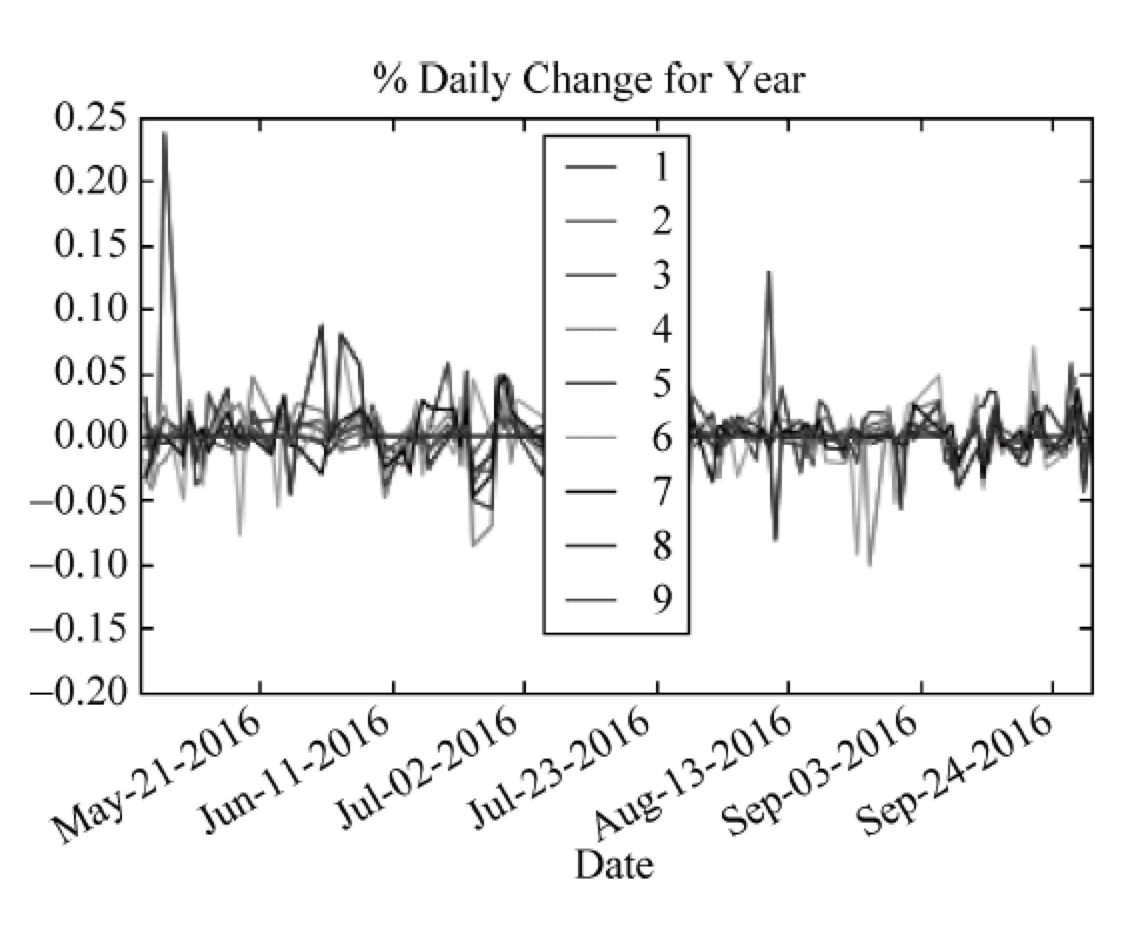


图 13-3　投资组合相关性的年度线性图

得到如图 13-4 所示的图形。

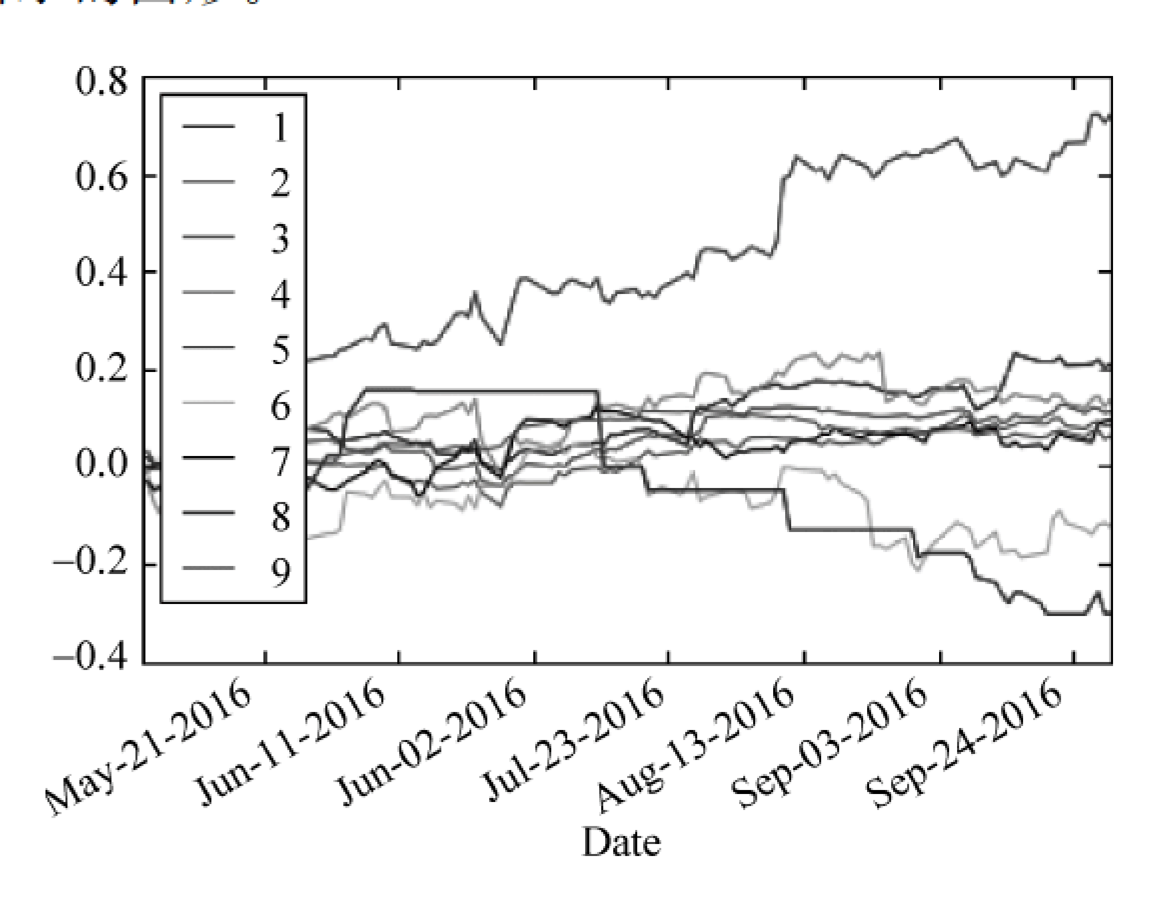


图 13-4　各资产收益率的累积和

图 13-4 所示的结果告诉了我们关于投资组合的更多信息。通过进行时间序列分析并绘制结果图标，SCPJ 显然面临着比原来想象的更加困难的时刻，年收益下降了近 40%，9 月甚至一度下降了 40%。有关股票走势的其他数据表明，SCPJ 的标准偏差相当高。因为标准偏差是风险的大致表现，所以，在制订该组合并确定权重时，应重点关注这个地方。

13.1.7　Pandas 组合相关性的百分比变化

确定九种股票间百分比变化的相关性与调用 DataFrame 收益 corr 的方法一样简单：

```
returns.corr()
Out[21]:
          1         2         3         4         5         6         7  \
1  1.000000  0.416977  0.441595  0.359194  0.117748  0.076796  0.201243
2  0.416977  1.000000  0.615684  0.431009  0.379273  0.061064  0.336605
3  0.441595  0.615684  1.000000  0.441726  0.373359  0.068465  0.351125
4  0.359194  0.431009  0.441726  1.000000  0.274183  0.007723  0.460048
5  0.117748  0.379273  0.373359  0.274183  1.000000  0.194522  0.222843
6  0.076796  0.061064  0.068465  0.007723  0.194522  1.000000  0.203828
```

```
7  0.201243  0.336605  0.351125  0.460048  0.222843  0.203828  1.000000
8  0.089218  0.077398  0.082808  0.081331  0.182187  0.175279  0.071175
9  0.261049  0.440014  0.534019  0.265154  0.199166  0.151390  0.332621
          8         9
1  0.089218  0.261049
2  0.077398  0.440014
3  0.082808  0.534019
4  0.081331  0.265154
5  0.182187  0.199166
6  0.175279  0.151390
7  0.071175  0.332621
8  1.000000  0.144818
9  0.144818  1.000000
```

13.1.8 SPY 标准普尔 500 指数的累积收益率图

在下面的标准普尔 500 的实例中，创建了另一个 DataFrame，在同一时间周期内，它可以充当“市场投资组合”。在图 13-5 中的图表展示了 SPY 生成的收益率，SPY 是标准普尔 500 指数的代理。

```
market_data_feed = {}
market_data_feed[1] = web.get_data_yahoo('SPY', '05/1/2016', '10/1/2016')
market_symbols = ['SPY']
for ticker in market_symbols:
  market_data_feed[ticker] = web.get_data_yahoo (ticker, '05/1/2016', '10/1/2016')
market_price = DataFrame({tic: data['Adj Close'] for tic, data in market_data_feed.iteritems()})
market_volume = DataFrame({tic: data['Volume']for tic, data in market_data_feed.iteritems()})
#收盘价转为收益率
market_returns = market_price.pct_change()
#累积收益率
market_returns.cumsum()
mts = market_returns.cumsum()
plt.figure(); mts.plot(); plt.legend(loc = 'upper left')
plt.show()
```

得到如图 13-5 所示的图形。

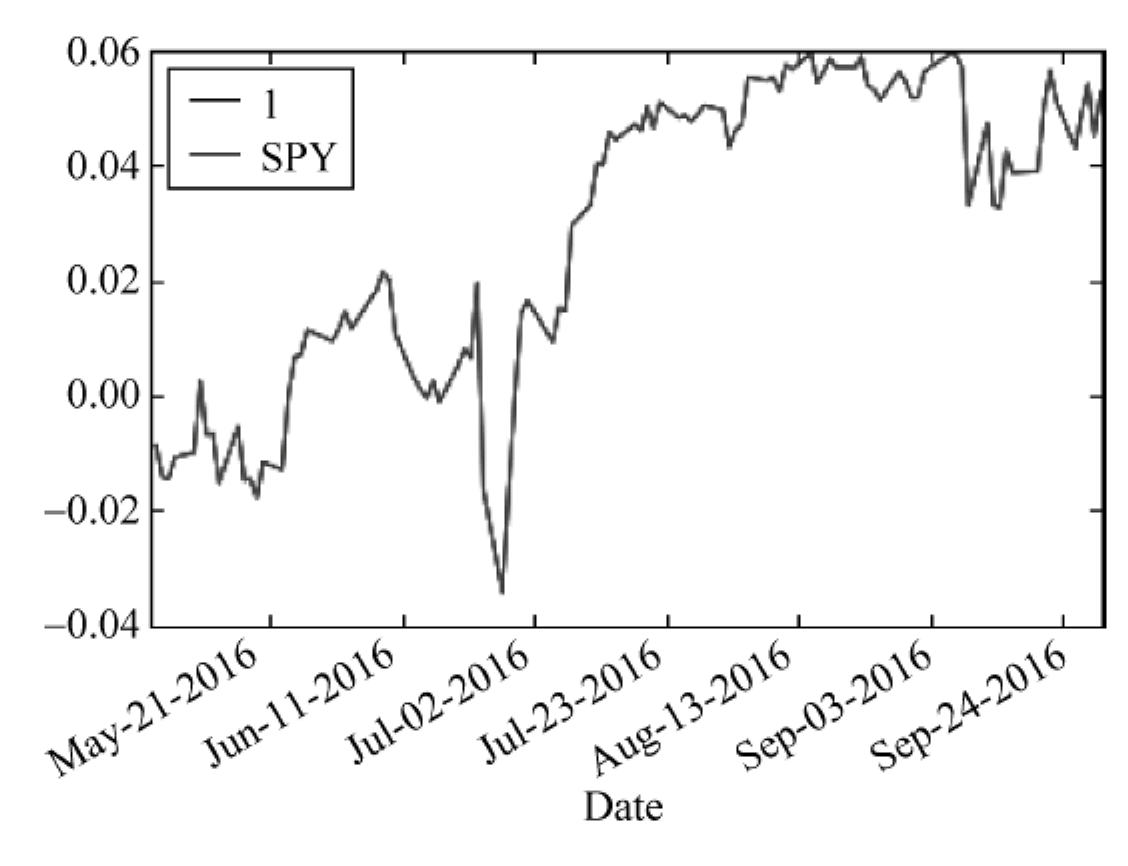
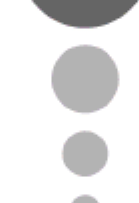


图 13-5 标准普尔 500 指数收益率的累积和



13.1.9　战胜股票市场的数据展示

在完成两个时间序列的图表后，下一步是分析查看与市场投资组合相对的产品投资组合。两种临时应急的方法是：(1)查看你的组合与市场投资组合的平均收益率；(2)查看标准差(stdev)，这是一种关于上述投资组合与市场投资组合的大致风险代理。

```
sum_returns = returns.sum()
sum_returns.mean()
Out[37]: 0.11059463984630956
market_returns.sum().mean()
Out[38]: 0.052580101035346494
market_returns.std()
Out[39]:
1        0.007627
SPY      0.007627
dtype: float64
returns.std().mean()
Out[40]: 0.01834054247037963
```

在最后交互实例中，可以通过 11.06%的投资组合收益率与 5.3%的市场投资组合收益率来战胜股市。在启动对冲基金之前，我们可能需了解为什么市场投资组合获得 0.76%的标准差，而我们的投资组合只获得了 1.83%的标准差。快速回答是，我们冒了较大风险，而且只是幸运罢了。进一步的分析涉及确定 alpha、beta、预期收益，以及进行 Fama-French 和有效边界优化之类的高级分析。

在本节中，Python 用于执行临时应急的投资组合分析。Python 逐渐变成用于真实数据分析的首选语言。Pandas(时间序列分析)、Pyomo(线性优化)、Numpy(数值计算)、Scipy(科学计算)和 IPython (Python 交互式计算和开发环境)之类的库使得在 Python 中应用高等数学知识变得更加轻松。

13.2　股票数据描述性统计的 Python 应用

13.2.1　程序包准备

```
#使用免费、开源的 Python 财经数据接口包 Tushare 来获取数据
import tushare as ts                 #需先安装 Tushare 程序包
#此程序包的安装是在 Anaconda Prompt 状态下，输入命令：pip install tushare
import Pandas as pd
import NumPy as np
import statsmodels.api as sm
import SciPy.stats as scs
import matplotlib.pyplot as plt
```

13.2.2　选择股票代号获取股票数据

```
symbols = ['hs300', '600000', '000980', '000981']
#把相对应股票的收盘价按照时间的顺序存入 DataFrame 对象中
```

```
data = pd.DataFrame()
hs300_data = ts.get_hist_data('hs300','2016-01-01','2016-12-31')
hs300_data = hs300_data['close']
hs300_data = hs300_data[::-1]
data['hs300'] = hs300_data
data1 = ts.get_hist_data('600000','2016-01-01','2016-12-31')
data1 = data1['close']
data1 = data1[::-1]
data['600000'] = data1
data2 = ts.get_hist_data('000980','2016-01-01','2016-12-31')
data2 = data2['close']
data2 = data2[::-1]
data['000980'] = data2
data3 = ts.get_hist_data('000981','2016-01-01','2016-12-31')
data3 = data3['close']
data3 = data3[::-1]
data['000981'] = data3
```

13.2.3 查看数据和清理数据

```
#查看股票收盘价
data.info()
<class 'Pandas.core.frame.DataFrame'>
Index: 184 entries, 2016-01-04 to 2016-09-30
Data columns (total 4 columns):
hs300      184 non-null float64
600000     166 non-null float64
000980     106 non-null float64
000981     137 non-null float64
dtypes: float64(4)
```

由上可见,各个股票的记录不一致。

```
#查看数据
data.head()
Out[40]:
                hs300    600000   000980   000981
date
2016-01-04   3469.066   17.73     NaN      NaN
2016-01-05   3478.780   17.96     NaN      NaN
2016-01-06   3539.808   18.11     NaN      NaN
2016-01-07   3294.384   17.63     NaN      NaN
2016-01-08   3361.563   17.49     NaN      NaN
```

从上可见 000980 与 000981 股票的记录有 null 值。

```
#数据清理
data = data.dropna()
```

13.2.4　股票数据的可视化

```
# 直接比较 4 只股票,但是规范化为起始值 100
(data / data.ix[0] * 100).plot(figsize = (8, 4))
```

得到如图 13-6 所示的图形。

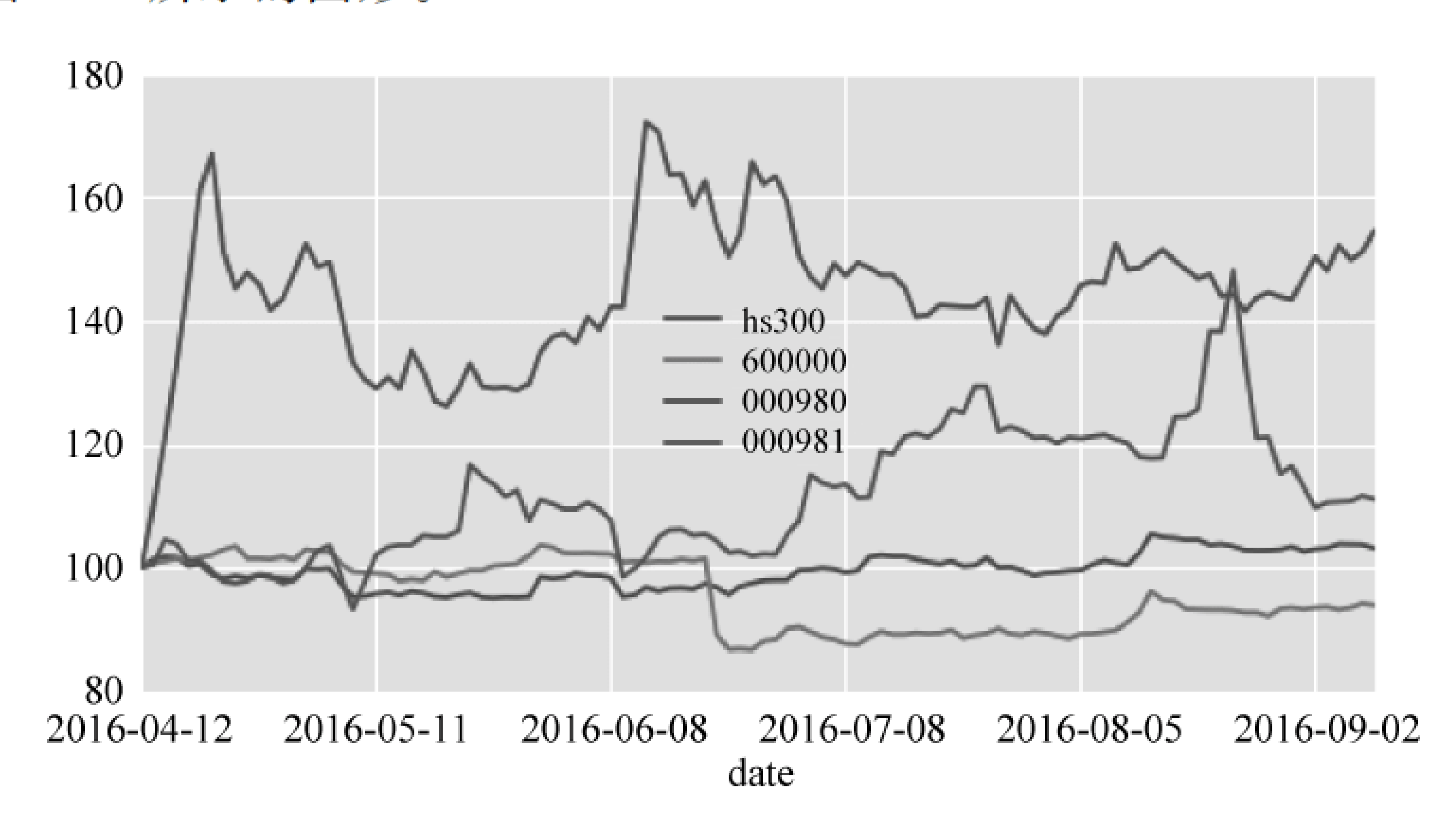
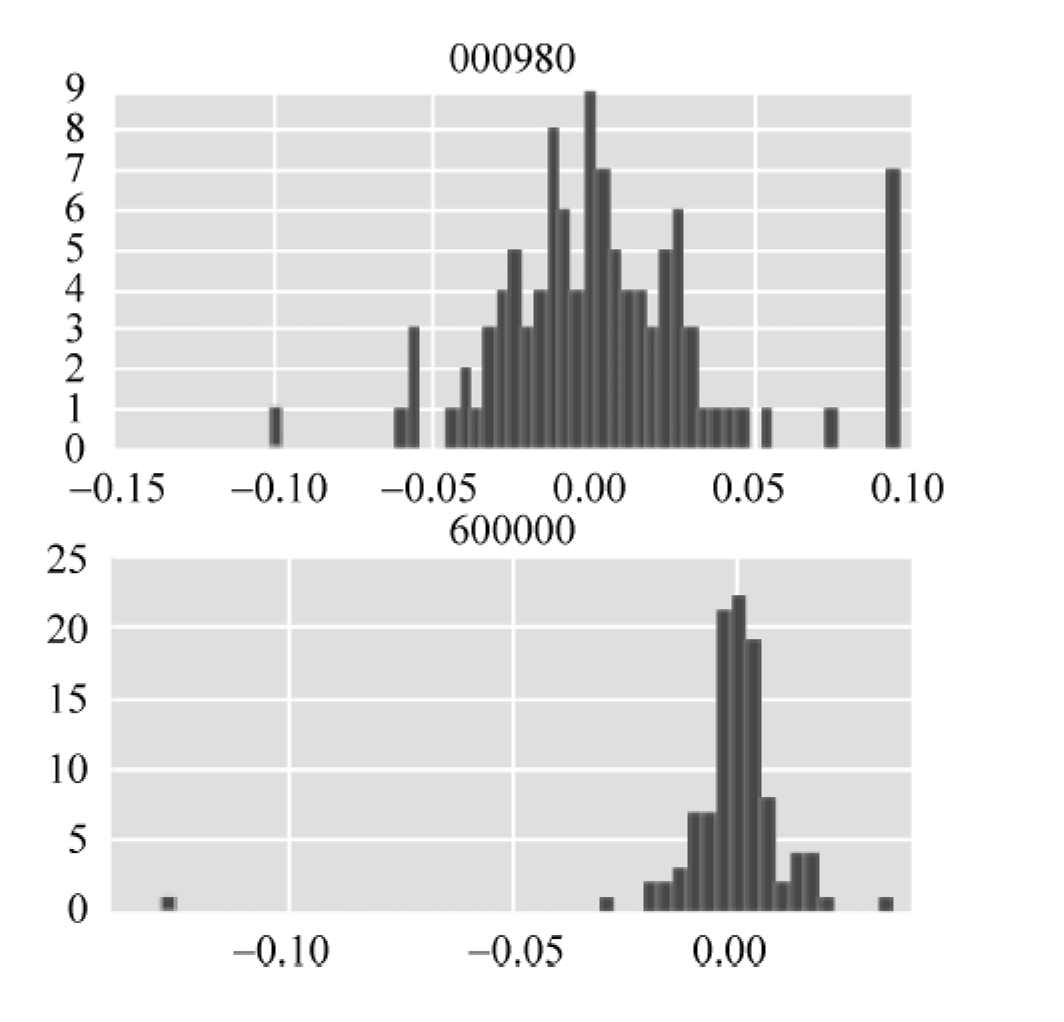


图 13-6　股票数据规范后价格变化图

```
# 用 Pandas 计算对数收益率比 Numpy 更方便一些,可使用 shift 方法:
log_returns = np.log(data / data.shift(1))
log_returns.head()
log_returns.hist(bins = 50, figsize = (9,4))
```

得到如图 13-7 所示的图形。



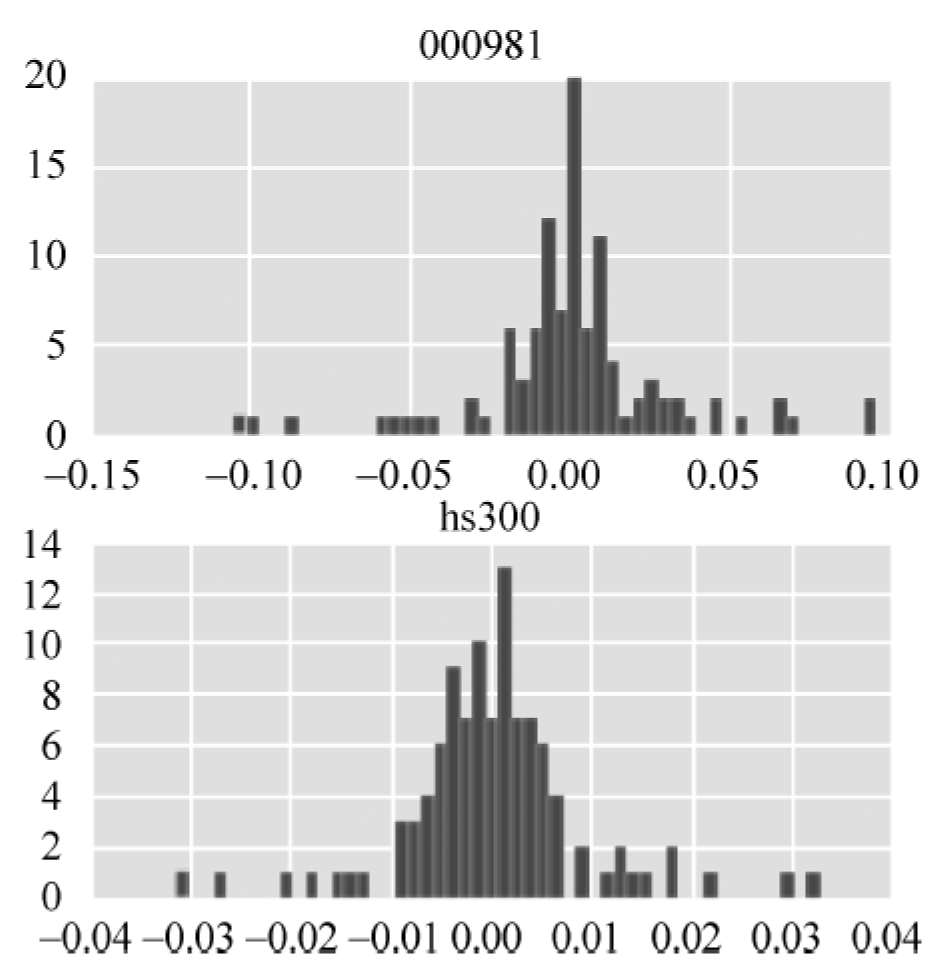


图 13-7　收益率的直方图

13.2.5　不同时间序列数据集的不同统计数值

下一步我们考虑时间序列数据集的不同统计数值。

```
# 峰度值似乎在所有 4 个数据集上都与正态分布的要求相差太多
```

```
#定义 print_statistics 函数,为了更加易于理解的方式
#输出给定(历史或者模拟)数据集均值、偏斜度或者峰度等统计数字
def print_statistics(array):
        sta = scs.describe(array)
        print "%14s %15s"        % ('statistic', 'value')
    print 30 * "-"
    print "%14s %15.5f" % ('size', sta[0])
    print "%14s %15.5f" % ('min', sta[1][0])
    print "%14s %15.5f" % ('max', sta[1][1])
    print "%14s %15.5f" % ('mean', sta[2])
    print "%14s %15.5f" % ('std', np.sqrt(sta[3]))
    print "%14s %15.5f" % ('skew', sta[4])
    print "%14s %15.5f" % ('kurtosis', sta[5])
for sym in symbols:
    print "\nResults for symbol %s" % sym
    print 30 * "-"
    log_data = np.array(log_returns[sym].dropna())
    print_statistics(log_data)
```

得到如下结果：

```
Results for symbol hs300
------------------------------
     statistic           value
------------------------------
          size       105.00000
           min        -0.03135
           max         0.03299
          mean         0.00029
           std         0.00912
          skew         0.20305
      kurtosis         3.30681

Results for symbol 600000
------------------------------
     statistic           value
------------------------------
          size       105.00000
           min        -0.12875
           max         0.03545
          mean        -0.00060
           std         0.01537
          skew        -5.46813
      kurtosis        44.53671

Results for symbol 000980
------------------------------
     statistic           value
------------------------------
          size       105.00000
           min        -0.10059
```

```
         max          0.09586
        mean          0.00415
         std          0.03576
        skew          0.68620
    kurtosis          1.37199

Results for symbol 000981
------------------------------
   statistic            value
------------------------------
        size        105.00000
         min         -0.10512
         max          0.09578
        mean          0.00100
         std          0.03058
        skew         -0.21642
    kurtosis          3.31592
```

13.2.6　通过分位数 qq 图检查代码的数据

```
#下面是 HS300 对数收益率 分位数 - 分位数图
sm.qqplot(log_returns['hs300'].dropna(), line='s')
plt.grid(True)
plt.xlabel('theoretical quantiles')
plt.ylabel('sample quantiles')
```

得到如图 13-8 所示的图形。

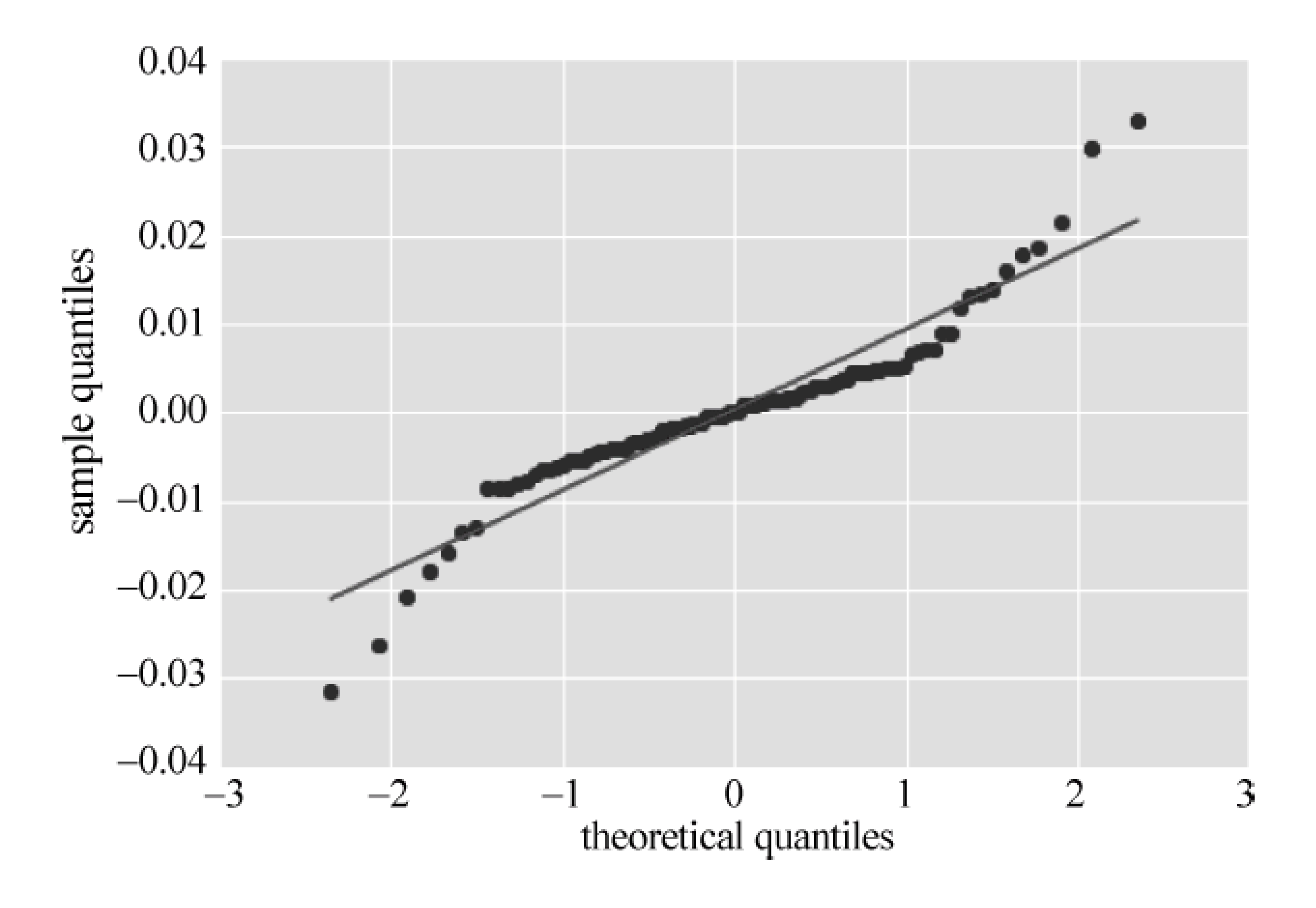


图 13-8　HS300 对数收益率分位数-分位数图

```
#从图 13-8 中可以看出：很显然,样本的分位数值不在一条直线上,表明"非正态性"。
#在左侧和右侧分别有许多值远低于和远高于直线。
#换句话说,这一时间序列信息展现出"大尾巴"(fat tails)。
#大尾巴指的是(频率)分布中观察到的正负异常值
#多于正态分布应有表现的情况。
#浦发银行 600000 对数收益率分位数 - 分位数图
sm.qqplot(log_returns['600000'].dropna(), line='s')
```

```
plt.grid(True)
plt.xlabel('theoretical quantiles')
plt.ylabel('sample quantiles')
```

得到如图 13-9 所示的图形。

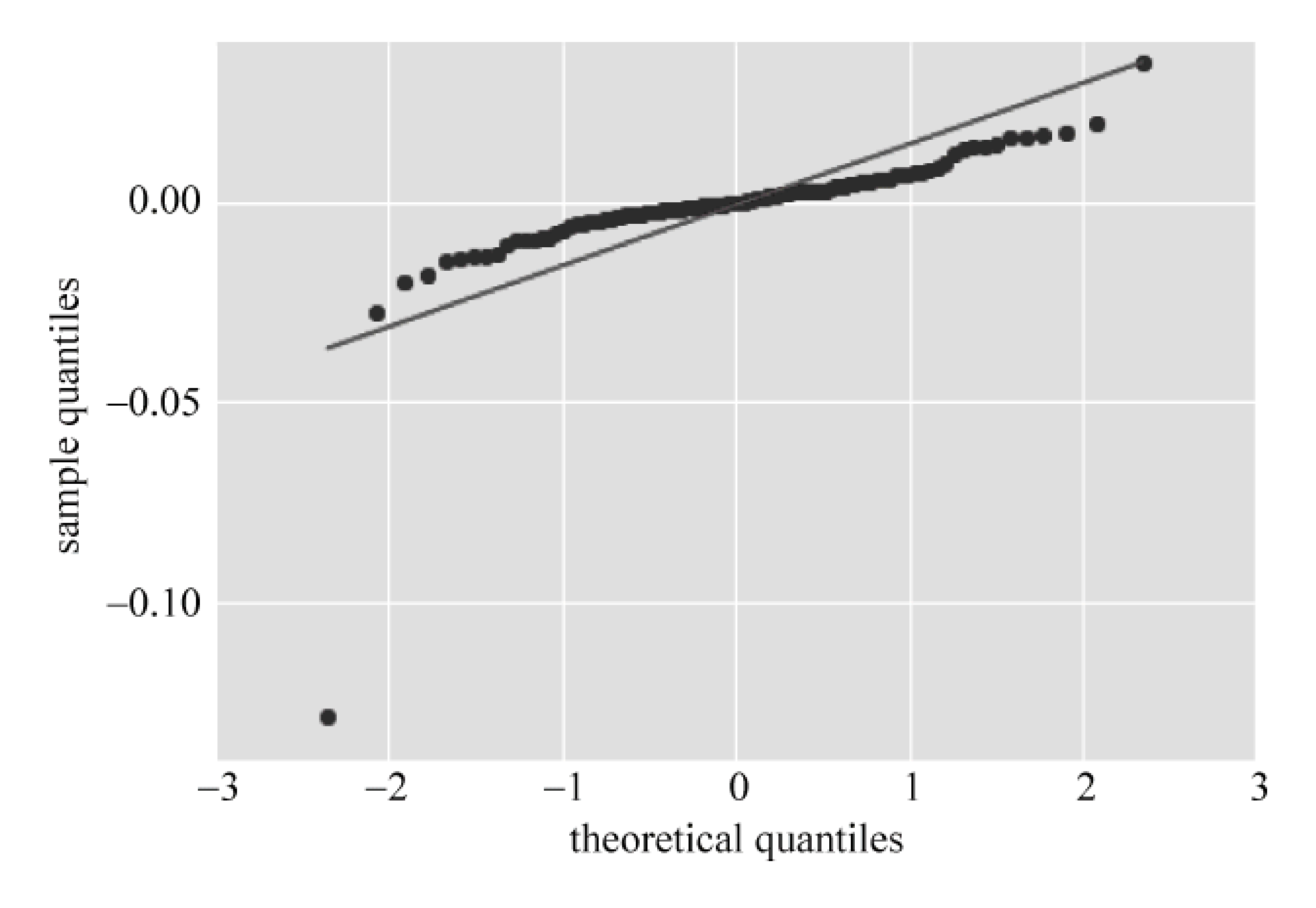


图 13-9 浦发银行 600000 对数收益率分位数-分位数图

13.2.7 正态性检验

```
def normality_tests(arr):
    '''Tests for normality distribution of given data set.
    normality_tests 函数组合了 3 中不同的统计学测试:
    偏斜度测试(Skewtest)
        测试样本数据的偏斜度是否"正态"(也就是值足够接近 0)
    峰度测试(kurtosistest)
        与上一种测试类似,测试样本数据的峰度是否"正态"(同样是值足够接近 0)
    正态性测试(normaltest)
        组合其他两种测试方法,检验正态性
    '''
    print "Skew of data set %14.3f" % scs.skew(arr)
    print "Skew test p-value %14.3f" % scs.skewtest(arr)[1]
    print "Kurt of data set %14.3f" % scs.kurtosis(arr)
    print "Kurt test p-value %14.3f" % scs.kurtosistest(arr)[1]
    print "Norm test p-value %14.3f" % scs.normaltest(arr)[1]
for sym in symbols:
    print "\nResults for symbol %s" % sym
    print 32 * "-"
    log_data = np.array(log_returns[sym].dropna())
    normality_tests(log_data)
```

得到如下结果:

```
Results for symbol hs300
--------------------------------
Skew of data set            0.203
Skew test p-value           0.371
```

```
Kurt of data set           3.307
Kurt test p-value          0.000
Norm test p-value          0.001

Results for symbol 600000
--------------------------
Skew of data set          -5.468
Skew test p-value          0.000
Kurt of data set          44.537
Kurt test p-value          0.000
Norm test p-value          0.000

Results for symbol 000980
--------------------------
Skew of data set           0.686
Skew test p-value          0.005
Kurt of data set           1.372
Kurt test p-value          0.020
Norm test p-value          0.001

Results for symbol 000981
--------------------------
Skew of data set          -0.216
Skew test p-value          0.341
Kurt of data set           3.316
Kurt test p-value          0.000
Norm test p-value          0.001
```

13.3　资产组合标准均值方差模型及其 Python 应用

本节先介绍资产组合均值方差模型要用到的一些概念，包括资产组合的可行集、资产组合的有效集、最优资产组合等。然后介绍标准均值方差模型及其应用。

13.3.1　资产组合的可行集

选择每个资产的投资比例，就确定了一个资产组合，在预期收益率 $E(r_P)$ 和标准差 σ_P 构成的坐标平面 $\sigma_P-E(r_P)$ 上就确定了一个点。因此，每个资产组合对应着 $\sigma_P-E(r_P)$ 坐标平面上的一个点；反之，$\sigma_P-E(r_P)$ 坐标平面上的一个点对应着某个特定的资产组合。如果投资者选择了所有可能的投资比例，则这些众多的资产组合点将在 $\sigma_P-E(r_P)$ 坐标平面上构成一个区域。这个区域称为资产组合的可行集或可行域。简言之，可行集是实际投资中所有可能的集合。也就是说，所有可能的组合将位于可行集的边界和内部。

13.3.2　有效边界与有效组合

1. 有效边界的定义

对于一个理性的投资者，他们都是厌恶风险而偏好收益的。在一定的收益下，他们将选

择风险最小的资产组合；在一定的风险下，他们将选择收益最大的资产组合。同时满足这两个条件的资产组合的集合就是有效集，又称有效边界。位于有效边界上的资产组合为有效组合。

2. 有效集的位置

有效集是可行集的一个子集。可行集、有效集、有效组合如图 13-10 所示。

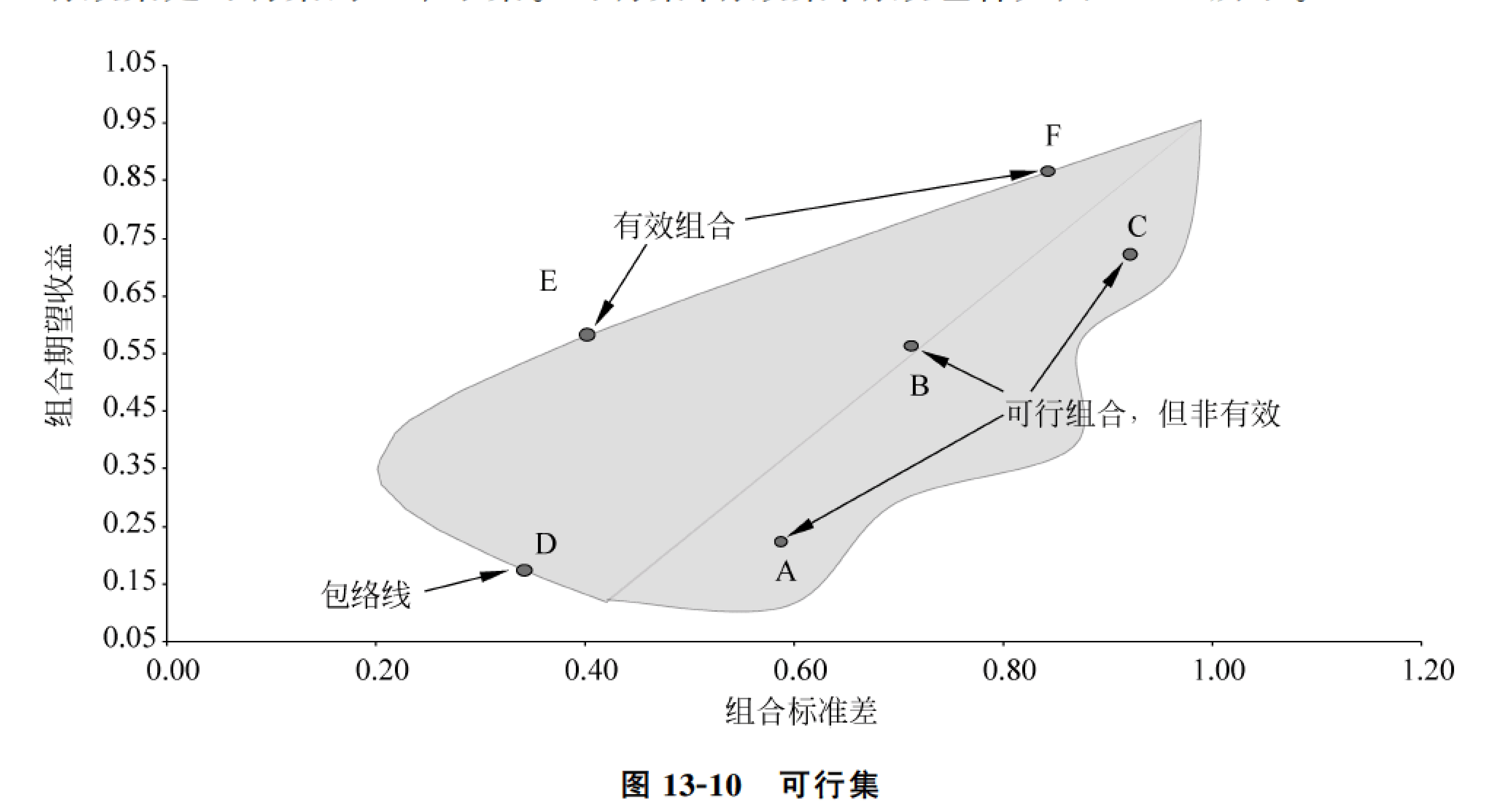


图 13-10 可行集

3. 最优资产组合的确定

在确定了有效集的形状之后，投资者就可以根据自己的无差异曲线选择效用最大化的资产组合。这个最优资产位于无差异曲线与有效集的相切点。

如图 13-11 所示，U_1，U_2，U_3 分别表示三条无差异曲线，它们的特点是下凸，其中 U_1 的效用水平最高，U_2 次之，U_3 最低，虽然投资者更加偏好于 U_1，但是在可行集上找不到这样的资产组合，因而是不可能实现的。U_3 上的资产组合虽然可以找到，但是由于 U_3 所代表的效用低于 U_2，所以 U_3 上的资产组合都不是最优的资产组合。U_2 正好与有效边界相切，代表了可以实现的最高投资效用，因此 P 点所代表的组合就是最优资产组合。

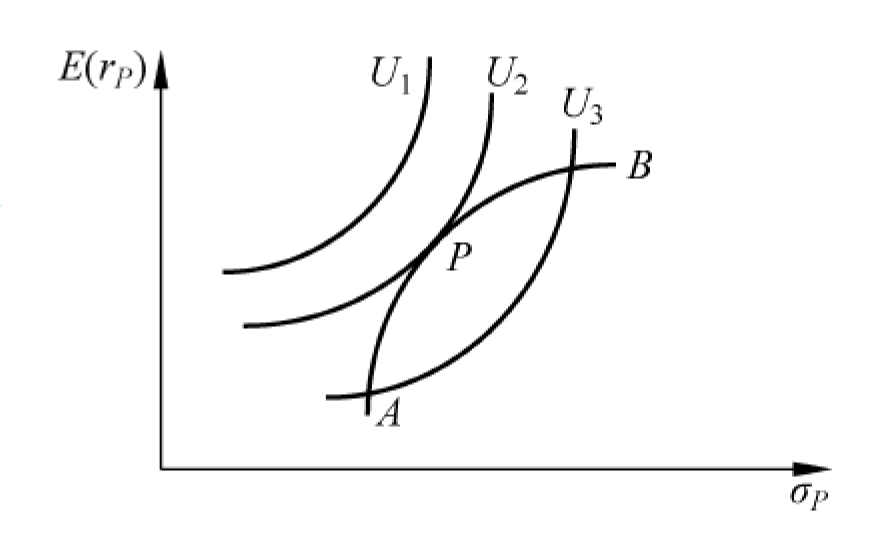


图 13-11 有效边界与无差异曲线

13.3.3 标准均值方差模型的求解

标准均值方差模型是标准的资产组合理论模型，也就是马科维茨最初创建的模型，它讨论的是理性投资者如何在投资收益风险两者之间进行权衡，以获得最优回报的问题。这个问题是一个二次规划问题，分为等式约束和不等式约束两种，我们只讨论等式约束下的资产组合优化问题。

在介绍资产组合理论之前，先引入如下概念。

定义：如果一个资产组合对确定的预期收益率有最小的方差，则称该资产组合为最小方差资产组合。

假设有 n 种风险资产，其预期收益率组成的向量记为 $\vec{e}=(E(r_1),E(r_2),\cdots,E(r_n))^{\mathrm{T}}$，每种风险资产的权重向量是 $\boldsymbol{X}=(x_1,\cdots,x_n)^{\mathrm{T}}$，协方差矩阵记为 $\boldsymbol{V}=[\sigma_{ij}]_{n\times n}$，向量 $\vec{\mathbf{1}}=[1,1,\cdots,1]^{\mathrm{T}}$，并且假设协方差矩阵记为 $\boldsymbol{V}=[\sigma_{ij}]_{n\times n}$ 是非退化矩阵，$\vec{e}\neq k\vec{1}$（k 为任一常数）。相应地，该资产组合的收益率记为 $E(r_P)=\boldsymbol{X}^{\mathrm{T}}\vec{e}$，风险记为 $\sigma_P^2=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{X}$。

投资者的行为是：给定一定的资产组合预期收益率 μ 水平，选择资产组合使其风险最小。这其实就是要求解如下形式的问题（标准均值方差模型）：

$$\min \frac{1}{2}\sigma_P^2 = \frac{1}{2}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{X} \tag{1}$$

$$\text{s. t.} \begin{cases} \vec{\mathbf{1}}^{\mathrm{T}}\boldsymbol{X} = 1 \\ E(r_P) = \vec{e}^{\mathrm{T}}\boldsymbol{X} = \mu \end{cases}$$

这是一个等式约束的极值问题，我们可以构造 Lagrange 函数：

$$L(X,\lambda_1,\lambda_2) = \frac{1}{2}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{X} + \lambda_1(1-\vec{\mathbf{1}}^{\mathrm{T}}\boldsymbol{X}) + \lambda_2(\mu - \boldsymbol{X}^{\mathrm{T}}\vec{e}) \tag{2}$$

则最优的一阶条件为

$$\begin{aligned} \frac{\partial L}{\partial X} &= \boldsymbol{V}\boldsymbol{X} - \lambda_1\vec{\mathbf{1}} - \lambda_2\vec{e} = \vec{\mathbf{0}} \\ \frac{\partial L}{\partial \lambda_1} &= 1 - \vec{\mathbf{1}}\boldsymbol{X} = 0 \\ \frac{\partial L}{\partial \lambda_2} &= \mu - \vec{e}^{\mathrm{T}}\boldsymbol{X} \end{aligned} \tag{3}$$

由(3)得最优解：

$$\boldsymbol{X} = \boldsymbol{V}^{-1}(\lambda_1\vec{\mathbf{1}} + \lambda_2\vec{e}) \tag{4}$$

(4) 分别左乘 $\vec{\mathbf{1}}^{\mathrm{T}}$ 和 $\vec{e}^{\mathrm{T}}$ 得

$$\begin{cases} 1 = \lambda_1\vec{\mathbf{1}}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{1} + \lambda_2\vec{\mathbf{1}}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{e} = \lambda_1 a + \lambda_2 b \\ \mu = \lambda_1\vec{e}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{1} + \lambda_2\vec{e}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{e} = \lambda_1 b + \lambda_2 c \end{cases} \tag{5}$$

$$\text{记} \begin{cases} a = \vec{\mathbf{1}}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{\mathbf{1}} \\ b = \vec{\mathbf{1}}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{e} \\ c = \vec{e}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{e} \\ \Delta = ac - b^2 \end{cases}$$

从而方程组(5)有解。如果 $\vec{e}\neq k\vec{\mathbf{1}}$，则 $\Delta=0$，此时除 $\mu=k$ 外，方程无解。解 λ_1,λ_2 方程组(5)得

$$\begin{cases} \lambda_1 = (c-\mu b)/\Delta \\ \lambda_2 = (\mu a - b)/\Delta \end{cases} \tag{6}$$

将(6)代入(4)得

$$\boldsymbol{X}=\boldsymbol{V}^{-1}\left(\frac{(c-\mu b)\vec{\mathbf{1}}}{\Delta}+\frac{(\mu a-b)\vec{\boldsymbol{e}}}{\Delta}\right)=\frac{\boldsymbol{V}^{-1}(c-\mu b)\vec{\mathbf{1}}}{\Delta}+\frac{\boldsymbol{V}^{-1}(\mu a-b)\vec{\boldsymbol{e}}}{\Delta}$$

$$=\frac{\boldsymbol{V}^{-1}(c\vec{\mathbf{1}}-b\vec{\boldsymbol{e}})}{\Delta}+\mu\frac{\boldsymbol{V}^{-1}(a\vec{\boldsymbol{e}}-b\vec{\mathbf{1}})}{\Delta} \tag{7}$$

再将(4)代入(2)得到最小方差资产组合的方差

$$\sigma_P^2=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{X}=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{V}^{-1}(\lambda_1\vec{\mathbf{1}}+\lambda_2\vec{\boldsymbol{e}})=\boldsymbol{X}^{\mathrm{T}}(\lambda_1\vec{\mathbf{1}}+\lambda_2\vec{\boldsymbol{e}})=\lambda_1\boldsymbol{X}^{\mathrm{T}}\vec{\mathbf{1}}+\lambda_2\boldsymbol{X}^{\mathrm{T}}\vec{\boldsymbol{e}}$$

$$=\lambda_1+\lambda_2\mu=(a\mu^2-2b\mu+c)/\Delta \tag{8}$$

式(8)给出了资产组合权重与预期收益率的关系。根据(8)可知,最小方差资产组合在坐标平面 $\sigma(r_P)-E(r_P)$ 平面上有双曲线形式如图 13-12 所示。而在 $\sigma^2(r_P)-E(r_P)$ 平面上可有抛物线形式,如图 13-13 所示。

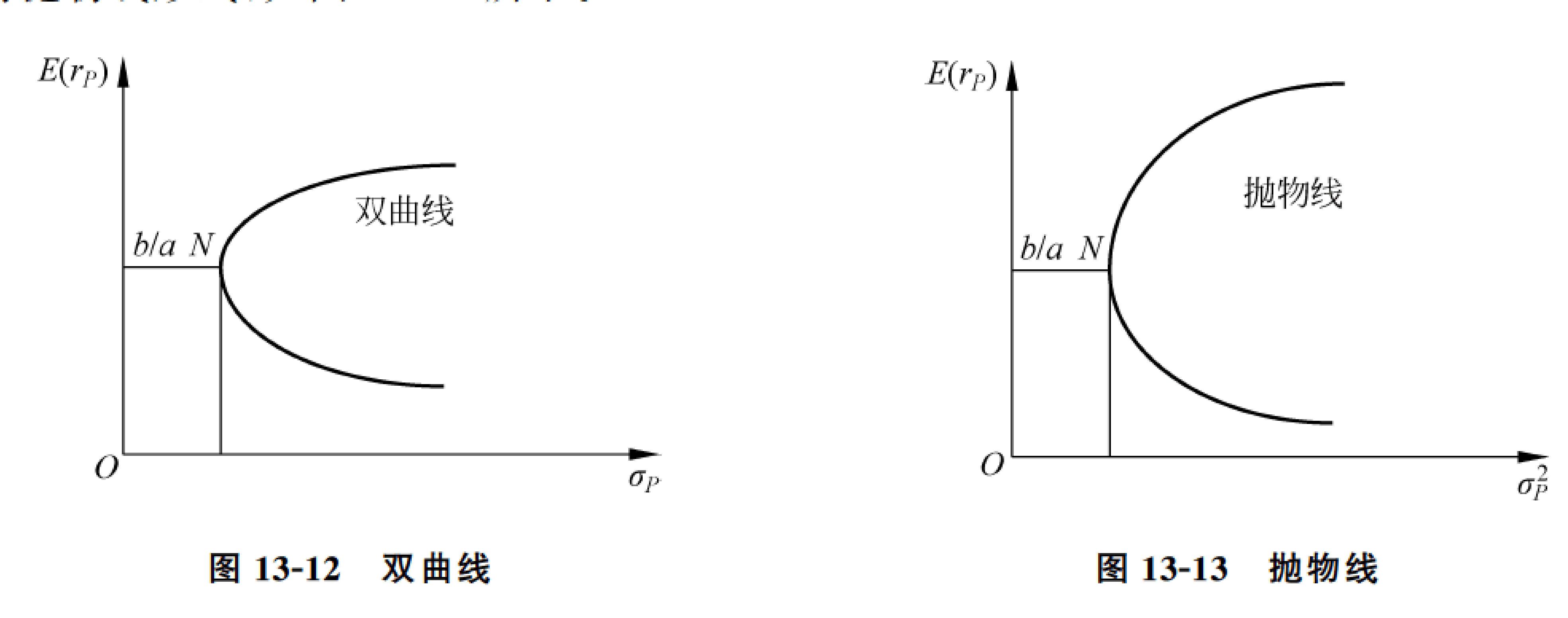


图 13-12 双曲线 **图 13-13 抛物线**

至此,我们得到描述最小方差资产组合的两个重要的量:

$$\boldsymbol{X}=\frac{\boldsymbol{V}^{-1}(c\vec{\mathbf{1}}-b\vec{\boldsymbol{e}})}{\Delta}+\mu\frac{\boldsymbol{V}^{-1}(a\vec{\boldsymbol{e}}-b\vec{\mathbf{1}})}{\Delta}$$

令 $\vec{\boldsymbol{g}}=\dfrac{\boldsymbol{V}^{-1}(c\vec{\mathbf{1}}-b\vec{\boldsymbol{e}})}{\Delta}$, $\vec{\boldsymbol{h}}=\dfrac{\boldsymbol{V}^{-1}(a\vec{\boldsymbol{e}}-b\vec{\mathbf{1}})}{\Delta}$,则 $\boldsymbol{X}=\vec{\boldsymbol{g}}+\mu\vec{\boldsymbol{h}}$

$$\sigma_P^2=(a\mu^2-2b\mu+c)/\Delta$$

13.3.4 标准均值方差模型的 Python 计算

例 13-1 考虑一个资产组合,其预期收益率矩阵为 $\vec{\boldsymbol{e}}=[0.05,0.1]^{\mathrm{T}}$,协方差矩阵是 $\boldsymbol{V}=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$,预期收益率 $\mu=0.075$,求最小方差资产组合的权重和方差。

解:$a=\vec{\mathbf{1}}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{\mathbf{1}}=[1\quad 1]\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}1\\ 1\end{bmatrix}$;$b=\vec{\mathbf{1}}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{\boldsymbol{e}}=[1\quad 1]\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}0.2\\ 0.5\end{bmatrix}$;

$$c=\vec{\boldsymbol{e}}^{\mathrm{T}}\boldsymbol{V}^{-1}\vec{\boldsymbol{e}}=[0.2\quad 0.5]\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}0.2\\ 0.5\end{bmatrix}$$

$$X=\vec{\boldsymbol{g}}+\mu\vec{\boldsymbol{h}};\quad \sigma_P^2=(a\mu^2-2b\mu+c)/\Delta$$

该实例计算的 Python 代码与计算结果如下:

```
from NumPy import *
v = mat('1 0;0 1')
print v
[[1 0]
[0 1]]
e = mat('0.05;0.1')
print e
[[ 0.05]
[ 0.1 ]]
ones = mat('1;1')
print ones
[[1]
[1]]
a = ones.T * v.I * ones
print a
[[ 2.]]
b = ones.T * v.I * e
print b
[[ 0.15]]
c = e.T * v.I * e
print c
[[ 0.0125]]
d = a * c - b * b
print d
[[ 0.0025]]
u = 0.075
c = 0.0125
b = 0.15
g = v.I * (c * ones - b * e)/d
a = 2.0
h = v.I * (a * e - b * ones)/d
x = g + h * u
print x
[[ 0.5]
[ 0.5]]
var = (a * u * u - 2 * b * u + c)/d
print var
[[ 0.5]]
```

13.4 资产组合有效边界的 Python 绘制

例 13-2 输入数据如表 13-1 所示。

表 13-1 已 知 数 据

输入各个证券的预期收益率				
	证券 1	证券 2	证券 3	证券 4
预期收益率	8%	12%	6%	18%
标准差	32%	26%	45%	36%

续表

输入各个证券间的协方差矩阵				
	证券 1	证券 2	证券 3	证券 4
证券 1	0.1024	0.0328	0.0655	−0.0022
证券 2	0.0328	0.0676	−0.0058	0.0184
证券 3	0.0655	−0.0058	0.2025	0.0823
证券 4	−0.0022	0.0184	0.0823	0.1296
输入单位向量转置	1	1	1	1

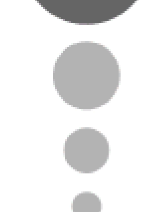
建立 Excel 数据文件为 yxbj.xls,数据如下:

u
0.01
0.03
0.05
0.07
0.09
…
0.35
0.37
0.39

利用上述给出的数据,绘制四个资产组成的投资组合的有效边界。

为了绘制四个资产投资组合的有效边界,我们编制 Python 代码如下:

```
import Pandas as pd
import NumPy as np
import matplotlib.pyplot as plt #绘图工具
#读取数据并创建数据表,名称为 u
u = pd.DataFrame(pd.read_excel('G:\\2glkx\\data\\yxbj.xls'))
V = mat('0.1024 0.0328 0.0655 - 0.0022;0.0328 0.0676 - 0.0058 0.0184;0.0655 - 0.0058 0.2025
0.0823; - 0.0022 0.0184 0.0823 0.1296')
e = mat('0.08;0.12;0.06;0.18')
ones = mat('1;1;1;1')
a =  ones.T * V.I * ones
b =  ones.T * V.I * e
c =  e.T * V.I * e
d =  a * c - b * b
a = np.array(a)
b = np.array(b)
c = np.array(c)
d = np.array(d)
u = np.array(u)
```



```
var = (a * u * u - 2.0 * b * u + c)/d
sigp = sqrt(var)
print sigp,u
[[ 0.40336771]
 [ 0.35191492]
 [ 0.3043241 ]
 [ 0.2627026 ]
 [ 0.23030981]
 [ 0.21143086]
 [ 0.20974713]
 [ 0.22564387]
 [ 0.25586501]
 [ 0.29605591]
 [ 0.34272694]
 [ 0.39357954]
 [ 0.44718944]
 [ 0.50267524]
 [ 0.55947908]
 [ 0.61723718]
 [ 0.67570488]
 [ 0.73471279]
 [ 0.7941405 ]
 [ 0.85390036]]
[[ 0.01]
 [ 0.03]
 [ 0.05]
 [ 0.07]
 [ 0.09]
 [ 0.11]
 [ 0.13]
 [ 0.15]
 [ 0.17]
 [ 0.19]
 [ 0.21]
 [ 0.23]
 [ 0.25]
 [ 0.27]
 [ 0.29]
 [ 0.31]
 [ 0.33]
 [ 0.35]
 [ 0.37]
 [ 0.39]]
plt.plot(sigp, u,'ro')
```

用 sigp 和 u 的数据可得到如图 13-14 所示 4 个资产投资组合的有效边界。

从上面显示的数据和图 13-14 中，我们可以看出，最小风险（标准差）所对应的点是（0.20974713，0.13）。

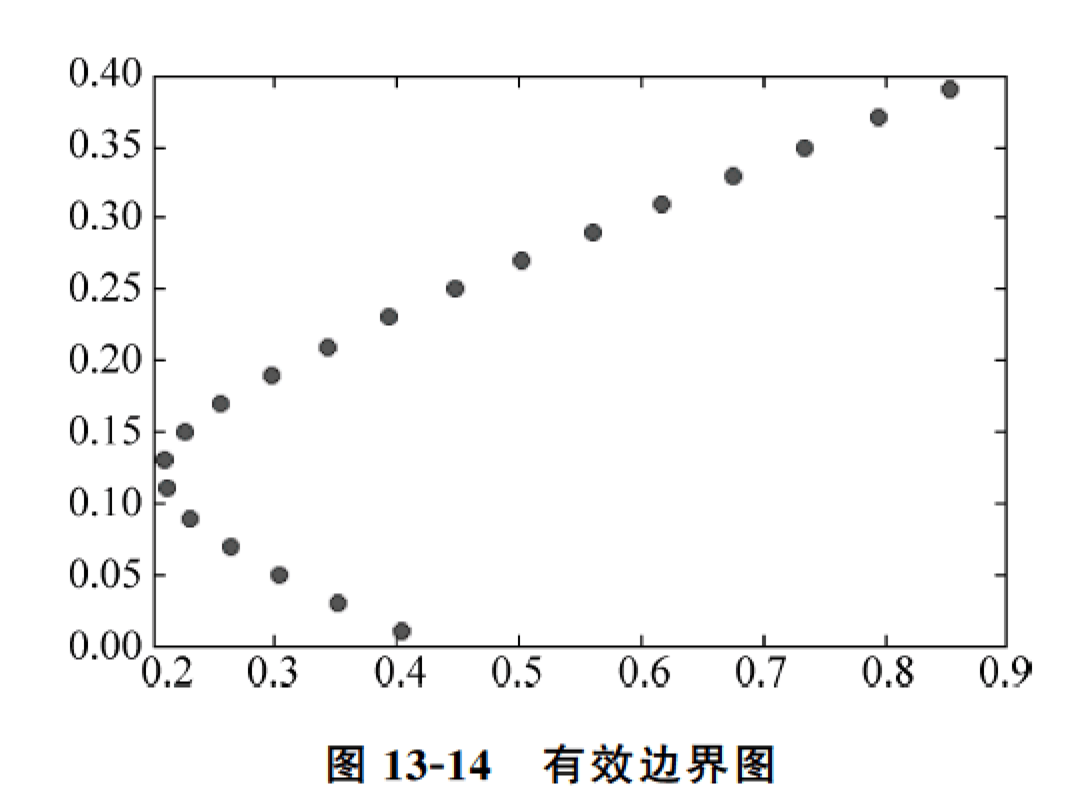


图 13-14　有效边界图

13.5　Markowitz 投资组合优化的 Python 应用

13.5.1　Markowitz 投资组合优化基本理论

多股票策略回测时常常遇到这样的问题：仓位如何分配？其实，这个问题早在 1952 年马科维茨(Markowitz)就给出了答案，即：投资组合理论。根据这个理论，我们可以对多资产的组合配置进行三方面的优化。

(1) 找到有效边界(或有效前沿)，在既定的收益率下使投资组合的方差最小化；

(2) 找到 sharpe 最优的投资组合(收益-风险均衡点)；

(3) 找到风险最小的投资组合。

该理论基于用均值方差模型来表述投资组合的优劣的前提。我们将选取几只股票，用蒙特卡洛模拟来探究投资组合的有效边界。通过 Sharpe 比最大化和方差最小化两种优化方法来找到最优的投资组合配置权重参数。最后，刻画出可能的分布，两种最优以及组合的有效边界。

13.5.2　投资组合优化实例的 Python 应用

例 13-3　三个投资对象的单项回报率历史数据如表 13-2 所示。

表 13-2　三个投资对象的单项回报率历史数据

时期	股票 1	股票 2	债券
1	0	0.07	0.06
2	0.04	0.13	0.07
3	0.13	0.14	0.05
4	0.19	0.43	0.04
5	−0.15	0.67	0.07
6	−0.27	0.64	0.08
7	0.37	0	0.06
8	0.24	−0.22	0.04
9	−0.07	0.18	0.05
10	0.07	0.31	0.07

续表

时期	股票 1	股票 2	债券
11	0.19	0.59	0.1
12	0.33	0.99	0.11
13	−0.05	−0.25	0.15
14	0.22	0.04	0.11
15	0.23	−0.11	0.09
16	0.06	−0.15	0.1
17	0.32	−0.12	0.08
18	0.19	0.16	0.06
19	0.05	0.22	0.05
20	0.17	−0.02	0.07

求三个资产的投资组合夏普比最大化和方差最小化的权数。

先将此表数据在目录 G:\2glkx\data 下建立 tzsy.xls 数据文件。

```
#准备工作
import Pandas as pd
import NumPy as np                       #数值计算
import statsmodels.api as sm             #统计计算
import SciPy.stats as scs                #科学计算
import matplotlib.pyplot as plt          #绘制图形
```

1. 选择股票

```
#取数
data = pd.DataFrame()
data=pd.read_excel('F:\\2glkx\\data\\tzsy.xls')
data=pd.DataFrame(data)
#清理数据
data=data.dropna()
data.head()
data.plot(figsize = (8,3))
```

得到如图 13-15 所示的图形。

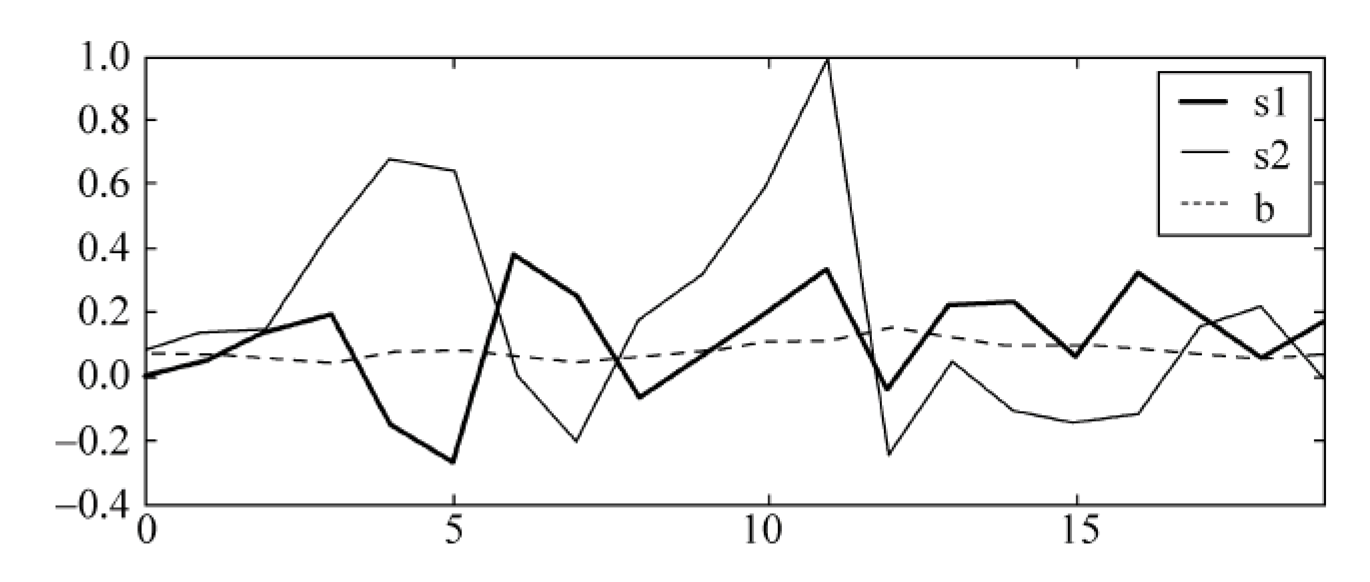


图 13-15　三个投资对象的收益率变化图

2. 计算不同证券的均值、协方差

```
returns = data
returns.mean()
Out[8]:
s1     0.1130
s2     0.1850
b      0.0755
dtype: float64
returns.cov()
Out[9]:
           s1          s2          b
s1   0.027433  -0.010768  -0.000133
s2  -0.010768   0.110153  -0.000124
b   -0.000133  -0.000124   0.000773
```

3. 给不同资产随机分配初始权重

```
noa = 3
weights = np.random.random(noa)
weights /= np.sum(weights)
weights
Out[10]: array([ 0.23377046, 0.51393812, 0.25229142])
```

4. 计算资产组合的预期收益、方差和标准差

```
np.sum(returns.mean() * weights)
Out[12]: 0.14054261642690027
np.dot(weights.T, np.dot(returns.cov(),weights))
Out[13]: 0.028007968937959201
np.sqrt(np.dot(weights.T, np.dot(returns.cov(),weights)))
Out[15]: 0.16735581536940747
```

5. 用蒙特卡洛模拟产生大量随机组合

给定的一个股票池(证券组合)如何找到风险和收益平衡的位置。下面通过一次蒙特卡洛模拟,产生大量随机的权重向量,并记录随机组合的预期收益和方差。

```
port_returns = []
port_variance = []
for p in range(4000):
  weights = np.random.random(noa)
  weights /= np.sum(weights)
  port_returns.append(np.sum(returns.mean() * weights))
  port_variance.append(np.sqrt(np.dot(weights.T, np.dot(returns.cov(), weights))))
port_returns = np.array(port_returns)
port_variance = np.array(port_variance)
#无风险利率设定为 4%
risk_free = 0.04
```

```
plt.figure(figsize = (8,3))
plt.scatter(port_variance, port_returns, c = (port_returns - risk_free)/port_variance, marker = 'o')
plt.grid(True)
plt.xlabel('excepted volatility')
plt.ylabel('excepted return')
plt.colorbar(label = 'Sharpe ratio')
```


得到如图 13-16 所示的图形。

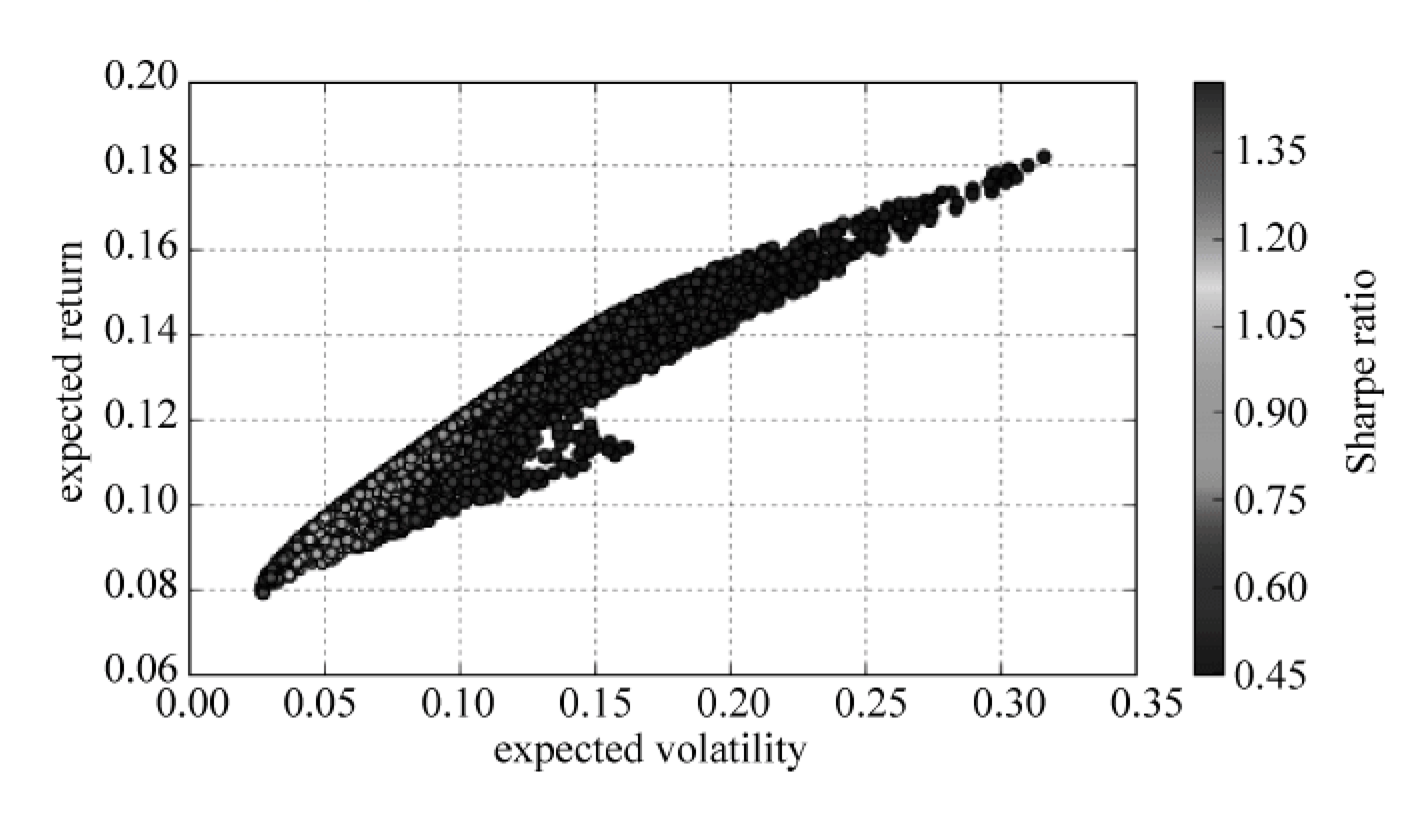


图 13-16　蒙特卡洛模拟产生大量随机投资组合

6. 投资组合优化 1——sharpe 最大

建立 statistics 函数来记录重要的投资组合统计数据(收益,方差和夏普比),通过对约束最优问题的求解,得到最优解。其中约束是权重总和为 1。

```
def statistics(weights):
    weights = np.array(weights)
    port_returns = np.sum(returns.mean() * weights)
    port_variance = np.sqrt(np.dot(weights.T, np.dot(returns.cov(),weights)))
    return np.array([port_returns, port_variance, port_returns/port_variance])
#最优化投资组合的推导是一个约束最优化问题
import SciPy.optimize as sco
#最小化夏普指数的负值
def min_sharpe(weights):
   return - statistics(weights)[2]
#约束是所有参数(权重)的总和为 1。这可以用 minimize 函数的约定表达如下
cons = ({'type':'eq', 'fun':lambda x: np.sum(x) - 1})
#我们还将参数值(权重)限制在 0 和 1 之间。这些值以多个元组组成的一个元组形式提供给最小化
函数
bnds = tuple((0,1) for x in range(noa))
#优化函数调用中忽略的唯一输入是起始参数列表(对权重的初始猜测)。我们简单的使用平均
分布。
opts = sco.minimize(min_sharpe, noa * [1./noa,], method = 'SLSQP', bounds = bnds, constraints =
cons)
opts
```

得到如下结果:

```
fun: -2.9195938061882454
     jac: array([ 0.01298031, -0.00767258, -0.00054446, 0. ])
 message: 'Optimization terminated successfully.'
    nfev: 44
     nit: 8
    njev: 8
  status: 0
 success: True
       x: array([ 0.05163244, 0.02181969, 0.92654787])
```

得到的最优组合权重向量为：

```
opts['x'].round(3)
Out[21]: array([ 0.052, 0.022, 0.927])
```

sharpe 最大的组合 3 个统计数据分别为：

```
#预期收益率、预期波动率、最优夏普指数
statistics(opts['x']).round(3)
Out[24]: array([ 0.08 , 0.027, 2.92 ])
```

7. 投资组合优化 2——方差最小

下面我们通过方差最小来选出最优投资组合：

```
def min_variance(weights):
   return statistics(weights)[1]
optv = sco.minimize(min_variance, noa*[1./noa,],method = 'SLSQP', bounds = bnds, constraints = cons)
optv
Out[25]:
     fun: 0.027037791350341657
     jac: array([ 0.0262073 , 0.02867849, 0.02704901, 0. ])
 message: 'Optimization terminated successfully.'
    nfev: 42
     nit: 8
    njev: 8
  status: 0
 success: True
       x: array([ 0.03570797, 0.01117468, 0.95311736])
```

方差最小的最优组合权重向量及组合的统计数据分别为：

```
optv['x'].round(3)
Out[26]: array([ 0.036, 0.011, 0.953])
#得到的预期收益率、波动率和夏普指数
statistics(optv['x']).round(3)
Out[27]: array([ 0.078, 0.027, 2.887])
```

8. 投资组合的有效边界

有效边界由既定的目标收益率下方差最小的投资组合构成。

在最优化时采用两个约束：(1)给定目标收益率；(2)投资组合权重和为 1。

```
  def min_variance(weights):
  return statistics(weights)[1]
#在不同目标收益率水平(target_returns)循环时,最小化的一个约束条件会变化.
target_returns = np.linspace(0.0,0.5,50)
target_variance = []
for tar in target_returns:
  cons = ({'type':'eq','fun':lambda x:statistics(x)[0] - tar},{'type':'eq','fun':lambda x:np.sum
(x) - 1})
  res = sco.minimize(min_variance, noa * [1./noa,], method = 'SLSQP', bounds = bnds,
constraints = cons)
  target_variance.append(res['fun'])
  target_variance = np.array(target_variance)
```

下面是最优化结果的展示。
叉号：构成的曲线是有效边界(目标收益率下最优的投资组合)
红星：sharpe 最大的投资组合
黄星：方差最小的投资组合

```
plt.figure(figsize = (8,3))
#圆圈:蒙特卡洛随机产生的组合分布
plt.scatter(port_variance, port_returns, c = port_returns/port_variance,marker = 'o')
#叉号:有效边界
plt.scatter(target_variance,target_returns, c = target_returns/target_variance, marker = 'x')
#红星:标记最高 sharpe 组合
plt.plot(statistics(opts['x'])[1], statistics(opts['x'])[0], 'r*', markersize = 15.0)
#黄星:标记最小方差组合
plt.plot(statistics(optv['x'])[1], statistics(optv['x'])[0], 'y*', markersize = 15.0)
plt.grid(True)
plt.xlabel('expected volatility')
plt.ylabel('expected return')
plt.colorbar(label = 'Sharpe ratio')
```

得到如图 13-17 所示的图形。

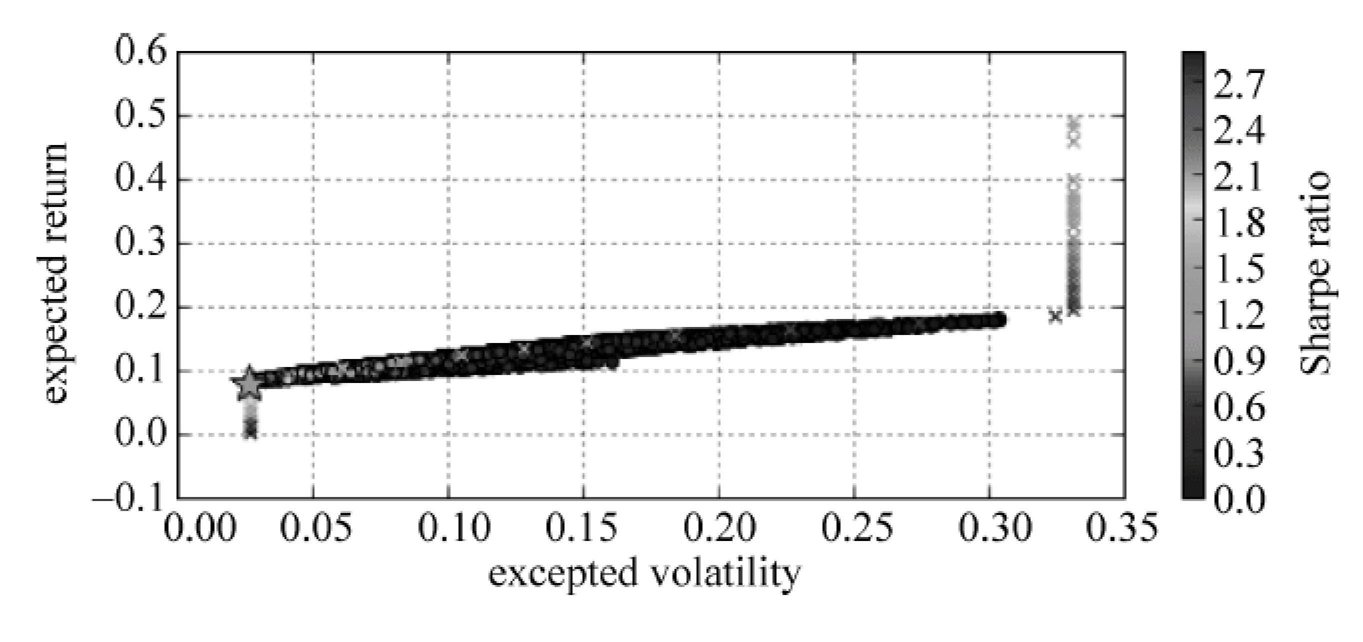


图 13-17　投资组合的有效边界

13.5.3 投资组合实际数据的 Python 应用

```
## 导入需要的程序包
import tushare as ts                    # 需先安装 tushare 程序包
# 此程序包的安装命令: pip install tushare
import Pandas as pd
import NumPy as np                      # 数值计算
import statsmodels.api as sm            # 统计运算
import SciPy.stats as scs               # 科学计算
import matplotlib.pyplot as plt         # 绘图
```

1. 选择股票代号、获取股票数据、清理及其可视化

```
symbols = ['hs300', '600000', '000980', '000981']
# 把相对应股票的收盘价按照时间的顺序存入 DataFrame 对象中
data = pd.DataFrame()
hs300_data = ts.get_hist_data('hs300','2016-01-01','2016-12-31')
hs300_data = hs300_data['close']        # 取沪深 300 收盘价数据
hs300_data = hs300_data[::-1]           # 按日期小到大排序
data['hs300'] = hs300_data
data1 = ts.get_hist_data('600000','2016-01-01','2016-12-31')
data1 = data1['close']                  # 浦发银行收盘价数据
data1 = data1[::-1]
data['600000'] = data1
data2 = ts.get_hist_data('000980','2016-01-01','2016-12-31')
data2 = data2['close']                  # 金马股份收盘价数据
data2 = data2[::-1]
data['000980'] = data2
data3 = ts.get_hist_data('000981','2016-01-01','2016-12-31')
data3 = data3['close']                  # 银亿股份收盘价数据
data3 = data3[::-1]
data['000981'] = data3
# 数据清理
data=data.dropna()
data.head()
# 规范化后时序数据
(data/data.ix[0]*100).plot(figsize = (8,4))
```

得到如图 13-18 所示的图形。

2. 计算不同证券的均值、协方差和相关系数

计算投资资产的协方差是构建资产组合过程的核心部分。运用 Pandas 内置方法生产协方差矩阵。

```
returns = np.log(data / data.shift(1))
returns.mean()*252
Out[63]:
hs300      0.073141
600000    -0.150356
```

13.5.3 投资组合实际数据的 Python 应用

```
## 导入需要的程序包
import tushare as ts                    #需先安装 tushare 程序包
#此程序包的安装命令: pip install tushare
import Pandas as pd
import NumPy as np                      #数值计算
import statsmodels.api as sm            #统计运算
import SciPy.stats as scs               #科学计算
import matplotlib.pyplot as plt         #绘图
```

1. 选择股票代号、获取股票数据、清理及其可视化

```
symbols = ['hs300', '600000', '000980', '000981']
#把相对应股票的收盘价按照时间的顺序存入 DataFrame 对象中
data = pd.DataFrame()
hs300_data = ts.get_hist_data('hs300','2016-01-01','2016-12-31')
hs300_data = hs300_data['close']        #取沪深 300 收盘价数据
hs300_data = hs300_data[::-1]           #按日期小到大排序
data['hs300'] = hs300_data
data1 = ts.get_hist_data('600000','2016-01-01','2016-12-31')
data1 = data1['close']                  #浦发银行收盘价数据
data1 = data1[::-1]
data['600000'] = data1
data2 = ts.get_hist_data('000980','2016-01-01','2016-12-31')
data2 = data2['close']                  #金马股份收盘价数据
data2 = data2[::-1]
data['000980'] = data2
data3 = ts.get_hist_data('000981','2016-01-01','2016-12-31')
data3 = data3['close']                  #银亿股份收盘价数据
data3 = data3[::-1]
data['000981'] = data3
#数据清理
data=data.dropna()
data.head()
#规范化后时序数据
(data/data.ix[0]*100).plot(figsize = (8,4))
```

得到如图 13-18 所示的图形。

2. 计算不同证券的均值、协方差和相关系数

计算投资资产的协方差是构建资产组合过程的核心部分。运用 Pandas 内置方法生产协方差矩阵。

```
returns = np.log(data / data.shift(1))
returns.mean()*252
Out[63]:
hs300       0.073141
600000     -0.150356
```

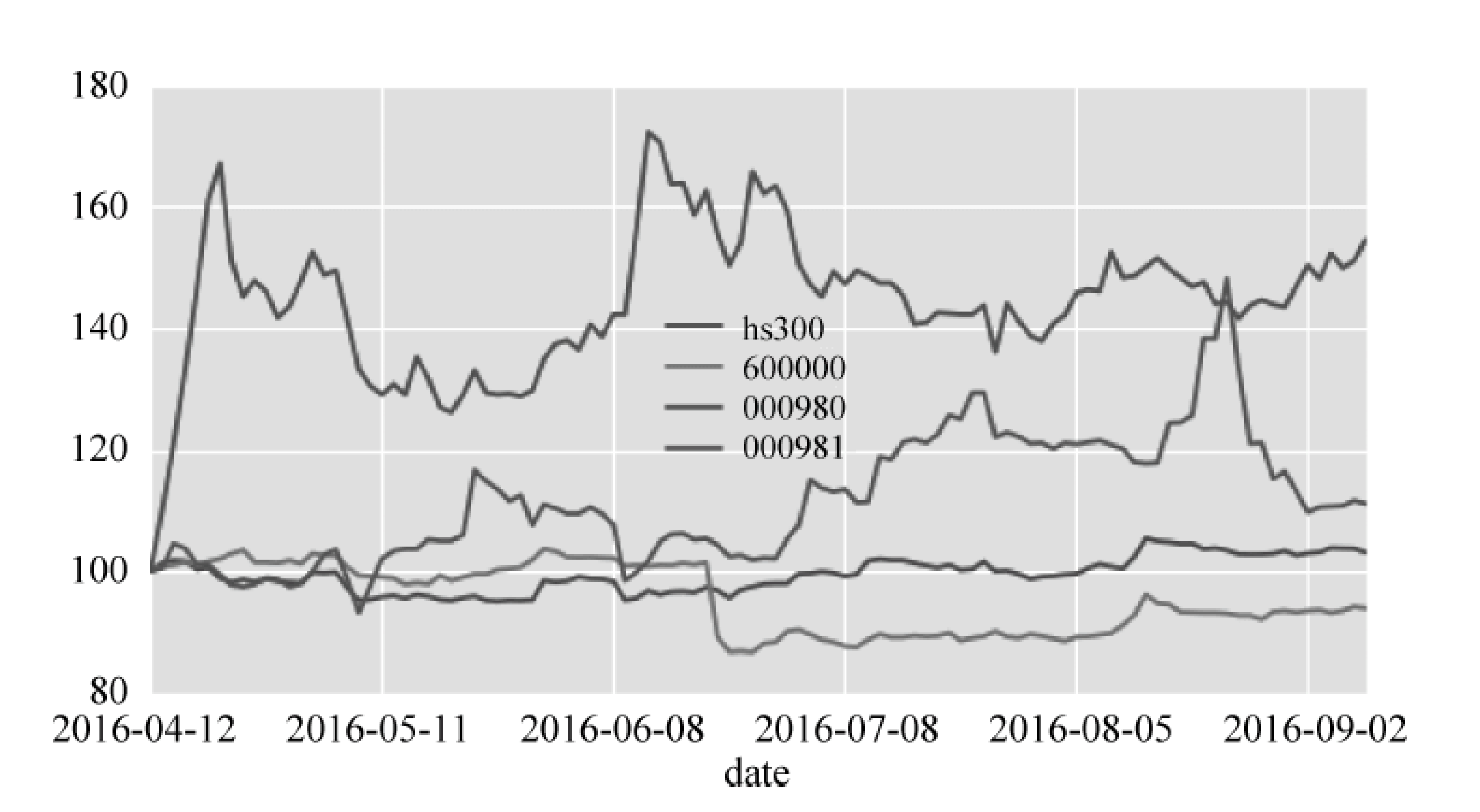


图 13-18　规范的时序价格变化

```
000980      1.044763
000981      0.252343
dtype: float64
returns.cov()                              #计算协方差
Out[64]:
              hs300     600000     000980     000981
hs300     0.000083  0.000051  0.000088  0.000095
600000    0.000051  0.000236  0.000081  0.000048
000980    0.000088  0.000081  0.001279  0.000111
000981    0.000095  0.000048  0.000111  0.000935
returns.corr()                             #计算相关系数
              hs300     600000     000980     000981
hs300     1.000000  0.363061  0.269357  0.341314
600000    0.363061  1.000000  0.146524  0.102416
000980    0.269357  0.146524  1.000000  0.101860
000981    0.341314  0.102416  0.101860  1.000000
```

由上可见，各证券之间的相关系数不太大，可以做投资组合。

3. 给不同资产随机分配初始权重

假设不允许建立空头头寸，所有的权重系数均在 0～1 之间。

```
noa = 4
weights = np.random.random(noa)
weights /= np.sum(weights)
weights
Out[65]: array([ 0.52080962, 0.33183961, 0.12028388, 0.02706689])
```

4. 计算预期组合收益、组合方差和组合标准差

```
np.sum(returns.mean() * weights)
Out[66]: 0.0004789557948133873
np.dot(weights.T, np.dot(returns.cov(),weights))
Out[67]: 0.00010701777937859502
np.sqrt(np.dot(weights.T, np.dot(returns.cov(),weights)))
Out[68]: 0.010344939795793644
```

5. 用蒙特卡洛模拟产生大量随机组合

现在,我们最想知道的是给定的一个股票池(投资组合),如何找到风险和收益平衡的位置。下面通过一次蒙特卡洛模拟,产生大量随机的权重向量,并记录随机组合的预期收益和方差。

```
port_returns = []
port_variance = []
for p in range(4000):
  weights = np.random.random(noa)
  weights /= np.sum(weights)
  port_returns.append(np.sum(returns.mean() * 252 * weights))
  port_variance.append(np.sqrt(np.dot(weights.T, np.dot(returns.cov() * 252, weights))))
port_returns = np.array(port_returns)
port_variance = np.array(port_variance)
#无风险利率设定为 1.5%
risk_free = 0.015
plt.figure(figsize = (8,4))
plt.scatter(port_variance, port_returns, c = (port_returns - risk_free)/port_variance, marker
 = 'o')
plt.grid(True)
plt.xlabel('expected volatility')
plt.ylabel('expected return')
plt.colorbar(label = 'Sharpe ratio')
```

得到如图 13-19 所示的图形。

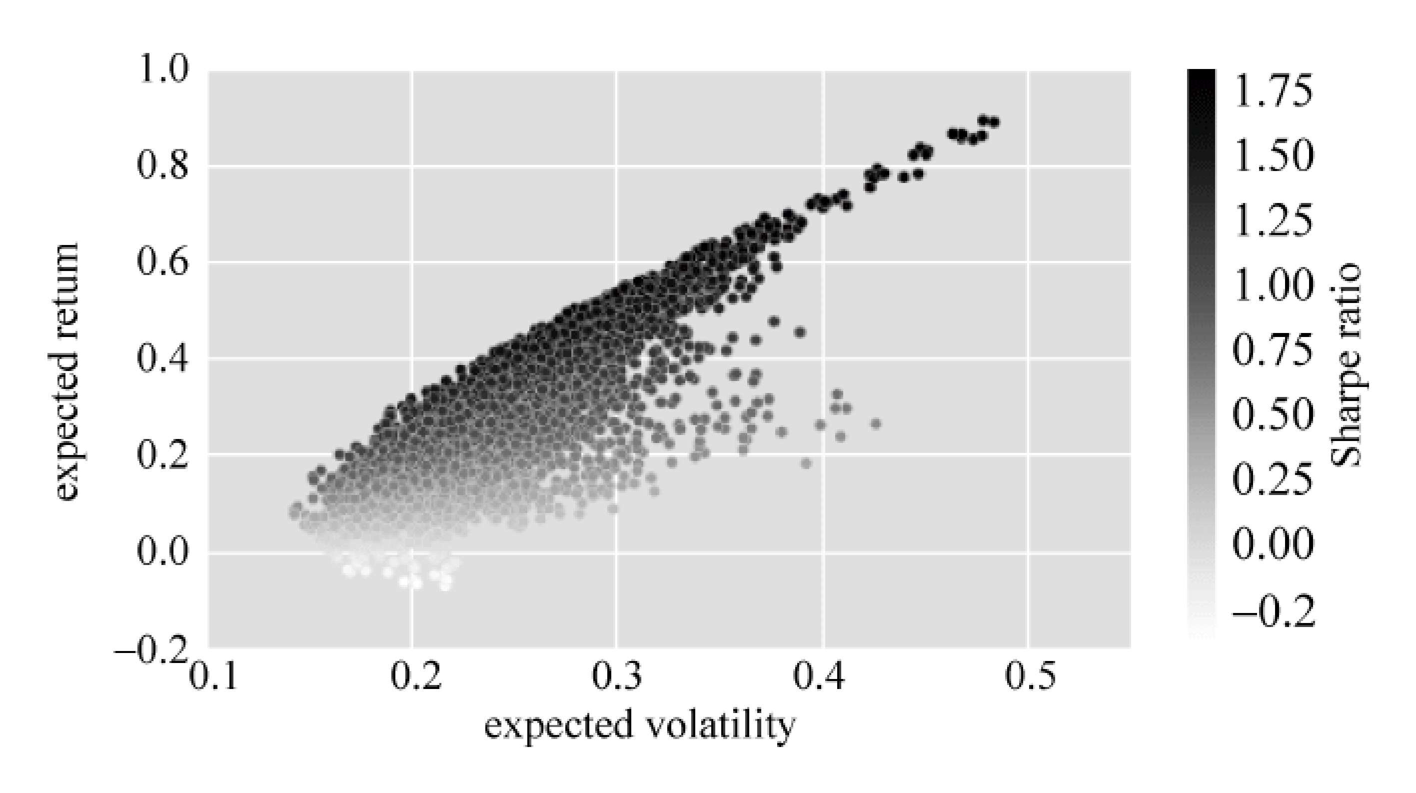


图 13-19 蒙特卡洛模拟产生大量随机组合

6. 投资组合优化 1——Sharpe 最大

建立 statistics 函数来记录重要的投资组合统计数据(收益、方差和夏普比)

通过对约束最优问题的求解,得到最优解。其中约束是权重总和为 1。

```
def statistics(weights):
   weights = np.array(weights)
   port_returns = np.sum(returns.mean() * weights) * 252
   port_variance = np.sqrt(np.dot(weights.T, np.dot(returns.cov() * 252,weights)))
```

5. 用蒙特卡洛模拟产生大量随机组合

现在,我们最想知道的是给定的一个股票池(投资组合),如何找到风险和收益平衡的位置。下面通过一次蒙特卡洛模拟,产生大量随机的权重向量,并记录随机组合的预期收益和方差。

```
port_returns = []
port_variance = []
for p in range(4000):
  weights = np.random.random(noa)
  weights /= np.sum(weights)
  port_returns.append(np.sum(returns.mean() * 252 * weights))
  port_variance.append(np.sqrt(np.dot(weights.T, np.dot(returns.cov() * 252, weights))))
port_returns = np.array(port_returns)
port_variance = np.array(port_variance)
#无风险利率设定为 1.5%
risk_free = 0.015
plt.figure(figsize = (8,4))
plt.scatter(port_variance, port_returns, c = (port_returns - risk_free)/port_variance, marker
 = 'o')
plt.grid(True)
plt.xlabel('expected volatility')
plt.ylabel('expected return')
plt.colorbar(label = 'Sharpe ratio')
```

得到如图 13-19 所示的图形。

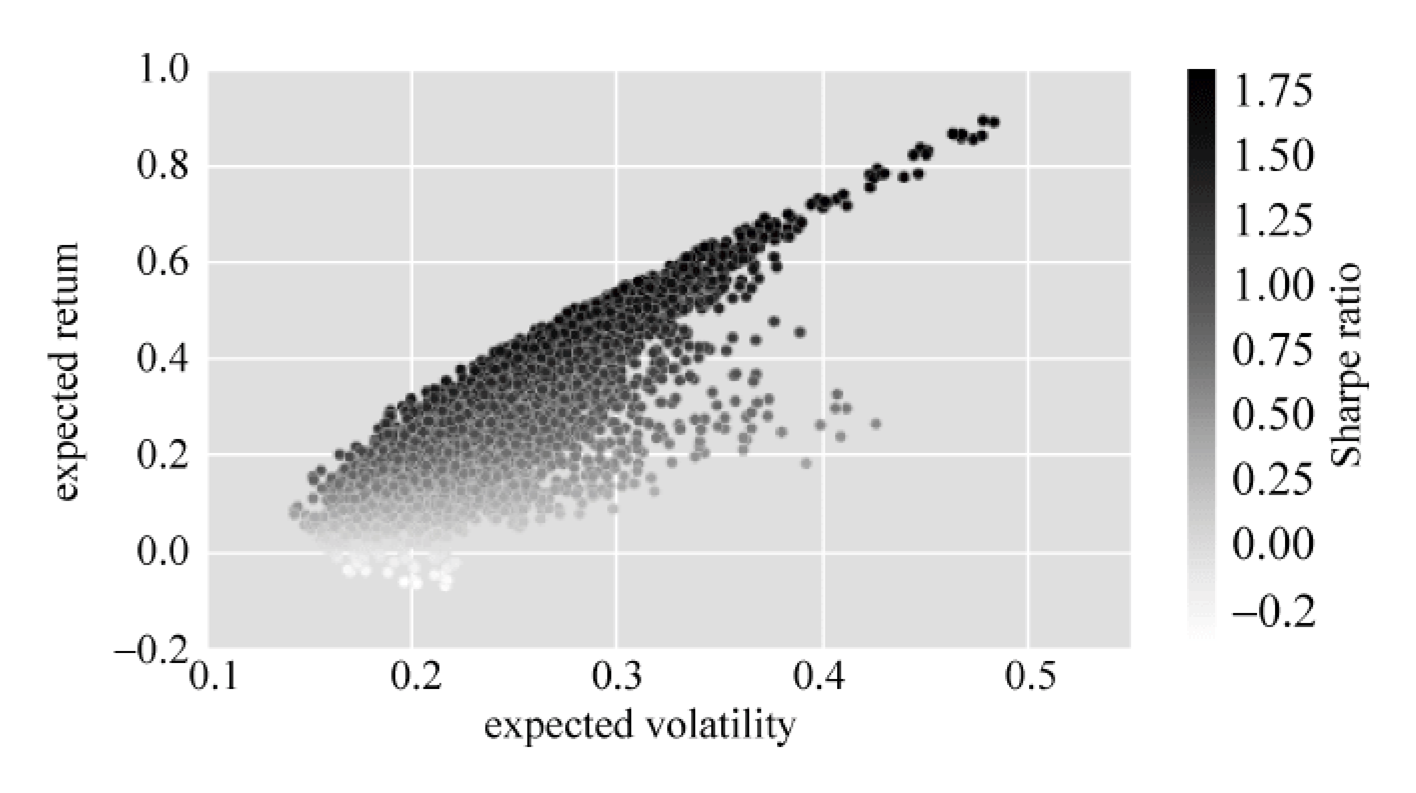


图 13-19 蒙特卡洛模拟产生大量随机组合

6. 投资组合优化 1——Sharpe 最大

建立 statistics 函数来记录重要的投资组合统计数据(收益、方差和夏普比)

通过对约束最优问题的求解,得到最优解。其中约束是权重总和为 1。

```
def statistics(weights):
   weights = np.array(weights)
   port_returns = np.sum(returns.mean() * weights) * 252
   port_variance = np.sqrt(np.dot(weights.T, np.dot(returns.cov() * 252,weights)))
```

```
    return np.array([port_returns, port_variance, port_returns/port_variance])
#最优化投资组合的推导是一个约束最优化问题
import SciPy.optimize as sco
#最小化夏普指数的负值
def min_Sharpe(weights):
  return - statistics(weights)[2]
#约束是所有参数(权重)的总和为 1。这可以用 minimize 函数的约定表达如下
cons = ({'type':'eq', 'fun':lambda x: np.sum(x) - 1})
#我们还将参数值(权重)限制在 0 和 1 之间。这些值以多个元组组成的一个元组形式提供给最小化
函数
bnds = tuple((0,1) for x in range(noa))
#优化函数调用中忽略的唯一输入是起始参数列表(对权重的初始猜测)。我们简单地使用平均
分布。
opts = sco.minimize(min_sharpe, noa * [1./noa,], method = 'SLSQP', bounds = bnds, constraints =
cons)
opts
```

运行上述代码,得到如下结果:

```
Out[90]:
     fun: - 1.870564674629059
     jac: array([ 2.87091583e - 02, 4.62549537e - 01, - 4.63277102e - 05,
         2.12848186e - 04, 0.00000000e + 00])
 message: 'Optimization terminated successfully.'
    nfev: 37
     nit: 6
    njev: 6
  status: 0
 success: True
       x: array([ 8.45677695e - 18, 0.00000000e + 00, 8.21263786e - 01,
         1.78736214e - 01])
```

输入如下代码:

```
opts['x'].round(3)
```

得到的最优组合权重向量为:

```
Out[91]: array([ 0.      ,    0.      ,    0.821,     0.179])
#预期收益率、预期波动率、最优夏普指数
statistics(opts['x']).round(3)
```

得到 Sharpe 最大的组合 3 个统计数据分别为:

```
Out[92]: array([ 0.903,  0.483,  1.871])
```

7. 投资组合优化 2——方差最小

下面我们通过方差最小来选出最优投资组合。

```
def min_variance(weights):
   return statistics(weights)[1]
optv = sco.minimize(min_variance, noa * [1./noa,],method = 'SLSQP', bounds = bnds, constraints =
```

```
cons)
optv
```

得到如下结果：

```
Out[94]:
     fun: 0.14048796305920866
     jac: array([ 0.14040739, 0.14094629, 0.15554342, 0.15803597, 0. ])
 message: 'Optimization terminated successfully.'
    nfev: 36
     nit: 6
    njev: 6
  status: 0
 success: True
       x: array([ 8.50485211e-01, 1.49514789e-01, 6.07153217e-18,
         6.07153217e-18])
```

方差最小的最优组合权重向量及组合的统计数据分别为：

```
optv['x'].round(3)
```

得到如下结果：

```
Out[95]: array([ 0.85,   0.15,   0.   ,   0.   ])
#得到的预期收益率、波动率和夏普指数
statistics(optv['x']).round(3)
```

得到如下结果：

```
Out[96]: array([ 0.04 , 0.14 , 0.283])
```

8. 投资组合的有效边界(前沿)

有效边界由既定的目标收益率下方差最小的投资组合构成。
在最优化时采用两个约束：(1)给定目标收益率；(2)投资组合权重和为 1。

```
def min_variance(weights):
   return statistics(weights)[1]
#在不同目标收益率水平(target_returns)循环时,最小化的一个约束条件会变化。
target_returns = np.linspace(0.0,0.5,50)
target_variance = []
for tar in target_returns:
  cons = ({'type':'eq','fun':lambda x:statistics(x)[0] - tar},{'type':'eq','fun':lambda x:np.sum
(x) - 1})
  res = sco.minimize(min_variance, noa * [1./noa,], method = 'SLSQP', bounds = bnds,
constraints = cons)
  target_variance.append(res['fun'])
target_variance = np.array(target_variance)
```

下面是最优化结果的展示。
叉号：构成的曲线是有效边界(目标收益率下最优的投资组合)

红星：Sharpe 最大的投资组合
黄星：方差最小的投资组合

```
plt.figure(figsize = (8,4))
#圆圈：蒙特卡洛随机产生的组合分布
plt.scatter(port_variance, port_returns, c = port_returns/port_variance,marker = 'o')
#叉号：有效边界
plt.scatter(target_variance,target_returns, c = target_returns/target_variance, marker = 'x')
#红星：标记最高 sharpe 组合
plt.plot(statistics(opts['x'])[1], statistics(opts['x'])[0], 'r*', markersize = 15.0)
#黄星：标记最小方差组合
plt.plot(statistics(optv['x'])[1], statistics(optv['x'])[0], 'y*', markersize = 15.0)
plt.grid(True)
plt.xlabel('expected volatility')
plt.ylabel('expected return')
plt.colorbar(label = 'Sharpe ratio')
```

得到如图 13-20 所示的图形。

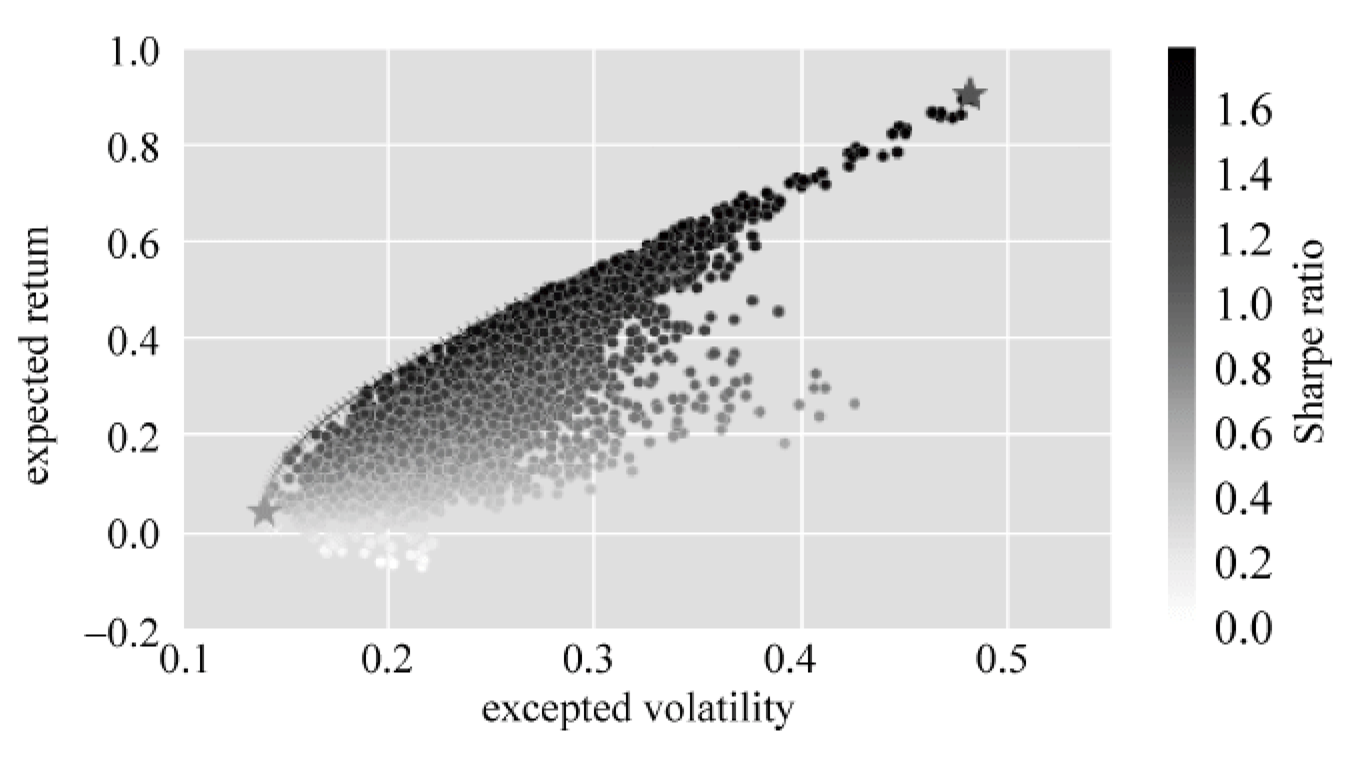


图 13-20 投资组合的可行集和有效边界

13.6 蒙特卡罗模拟股票期权定价的 Python 应用

设初始股票价格 S0=100，欧式看涨期权的执行价格 $X=100$，到期时间 $T=1$ 年，固定无风险利率 $r=10\%$；固定波动率 $\sigma=20\%$。我们有

$$S_T = S_0\exp[(r-0.5\sigma^2)T+\sigma\cdot z\sqrt{T}] \tag{1}$$

则计算欧式看涨期权的价格的步骤如下：

(1) 从标准正态分布中取得 i 个(伪)随机数 $z(i)$，$i\in\{1,2,\cdots,N\}$；

(2) 为给定的 z(i)和公式(1)计算到期的标的资产股票价格水平 $S_T(i)$；

(3) 计算到期时期权的所有内在价值 $h_T(i)=\max(S_T(i)-X,0)$，那么

$$C_0 = e^{-rT}\frac{1}{N}\sum_{i=1}^{N}h_T(i) \tag{2}$$

(4) 通过公式(2)中给出的蒙特卡洛方法估计的股票看涨期权现值。

例 13-4 考虑不支付红利股票的欧式看涨股票期权，它的标的资产股票价格是 100 元，执行价格是 100 元，无风险利率是 10%，年波动率是 25%，期权的有效期是 1 年，用蒙特卡罗法计算其欧式看涨股票期权的价格。

解：在本例中，$S_0=100.0$；$X=100.0$；$T=1.0$；$r=0.1$；sigma=0.25。

编制蒙特卡洛模拟法计算欧式看涨期权的 Python 代码如下：

```
S0 = 100.0;X = 100.0;T = 1.0;r = 0.1;sigma = 0.25
from NumPy import *
N = 50000
z = random.standard_normal(N)
ST = S0 * exp((r - 0.5 * sigma * sigma) * T + sigma * z * sqrt(T))
hT = maximum(ST - X,0)
C0 = exp( - r * T) * sum(hT)/N
print "C0 = ",C0
```

得到如下结果：

```
C0 =  14.9884387473
```

请读者思考：如果要求上面给定数据的欧式看跌股票期权，则上面的代码应如何修改？

我们把上面的程序代码编写成函数的形式如下：

```
def qqdj(S0,X,T,r,sigma,N):
  z = random.standard_normal(N)
  ST = S0 * exp((r - 0.5 * sigma * sigma) * T + sigma * z * sqrt(T))
  hT = maximum(ST - X,0)
  C0 = exp( - r * T) * sum(hT)/N
  return C0
from NumPy import *
res = qqdj(100,100,1.0,0.1,0.25,50000)
print "res = ",res
```

得到如下结果：

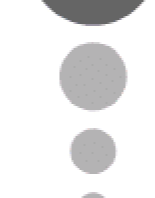
```
res =  14.9623748966
```

13.7 蒙特卡罗模拟期权价格稳定性的 Python 应用

蒙特卡罗模拟精度与模拟次数密切相关，模拟次数越多，其精度越高，但是次数增加又会增加计算量。实践表明，减少模拟方差可以提高稳定性，减少模拟次数。有很多方法可以减少方差，如对偶变量技术、控制变量技术、分层抽样、矩匹配、条件蒙特卡罗模拟等，但最简单且应用最为广泛的是对偶变量技术和控制变量技术。

13.7.1 对偶变量法

对偶变量技术就是先随机抽样得到一组数据，然后以此为基础构造出另一组对偶变量。下面以正态分布为例介绍对偶变量技术，首先从正态分布变量中随机抽取 N 个样本值，分别为 $Z_i(i=1,2,\cdots,N)$，由此可以得到 N 个模拟值 $C_i(i=1,2,\cdots,N)$，那么衍生证券蒙特卡



罗模拟估计值为

$$\hat{C}=\frac{1}{N}\sum_i C_i$$

以 $Z_i(i=1,2,\cdots,N)$为基础，构造对偶随机数$\widetilde{Z}_i=-Z_i$，$\widetilde{Z}_i$ 是与 $Z_i(i=1,2,\cdots,N)$相互对偶的随机数，由正态分布性质可知，$\widetilde{Z}_i=-Z_i(i=1,2,\cdots,N)$，也是服从正态分布，由对偶随机数生成的估计值为

$$\widetilde{C}=\frac{1}{N}\sum_i \widetilde{C}_i$$

对$\widetilde{C}$和$\hat{C}$取平均，得到新的估计值：

$$C=\frac{1}{2}(\widetilde{C}+\hat{C})=\frac{1}{N}\sum_i\left(\frac{\hat{C}_i+\widetilde{C}_i}{2}\right)$$

如果随机抽样的样本 $Z_i(i=1,2,\cdots,N)$模拟得到的估计值比较小，那么与之对偶的随机抽样样本$\widetilde{Z}_i=-Z_i(i=1,2,\cdots,N)$得到的估计值可能会偏大，二者的平均值就可能会接近真实值。如果 $\mathrm{cov}(\hat{C}_i,\widetilde{C}_i)\leqslant 0$，那么

$$\mathrm{var}\left(\frac{\hat{C}_i+\widetilde{C}_i}{2}\right)=\frac{1}{2}\mathrm{var}(\hat{C}_i)+\frac{1}{2}\mathrm{cov}(\hat{C}_i,\widetilde{C}_i)\leqslant\frac{1}{2}\mathrm{var}(\hat{C}_i)$$

从上面的不等式可以看出，利用对偶技术可以增加估计稳定性、提高估计精度。

根据对偶变量法的基本思想，编写 Python 程序的步骤大致如下：(1)模拟标的资产的价格路径；(2)计算两个期权损益值，其中一个是按照常规蒙特卡洛法计算的结果，另一个是改变所有正态分布符号计算出来的结果；(3)计算期权的价格，即计算上述两个期权的平均值并将计算结果进行贴现。

我们编制对偶变量法模拟期权价格的 Python 代码如下。

```
def doqqdj(S0,X,T,r,sigma,N):
    z = random.standard_normal(N)
    ST1 = S0 * exp((r - 0.5 * sigma * sigma) * T + sigma * z * sqrt(T))
    ST2 = S0 * exp((r - 0.5 * sigma * sigma) * T + sigma * ( - z) * sqrt(T))
    hT1 = maximum(ST1 - X,0)
    hT2 = maximum(ST2 - X,0)
    C1 = exp( - r * T) * sum(hT1)/N
    C2 = exp( - r * T) * sum(hT2)/N
    return (C1 + C2)/2
```

例 13-5　考虑不支付红利股票的欧式看涨期权，它们的标的资产价格是 100 元，执行价格是 100 元，无风险利率是 10%，年波动率是 25%，期权的有效期是 1 年，用对偶变量法计算其欧式看涨价格。

解：在本例中，$S=100$，$X=100$，$r=0.1$，$\sigma=0.25$，$T-t=1$。

利用对偶变量法计算期权价格的步骤如下。

(1) 模拟标的变量路径并计算$\hat{C}$

$$\begin{aligned}S_T^i&=S_t\exp[(r-0.5\sigma^2)(T-t)+\sigma\varepsilon\sqrt{T-t}]\\&=100\exp[(0.1-0.5\times0.25^2)\times1+0.25\varepsilon\sqrt{1}]\end{aligned}$$

$$\hat{C}=\max(0,S_T^i-X)$$

（2）改变随机变量 ε 的符号，模拟标的变量路径并计算 $\tilde{C}$

$$S_T^{i'}=S_t\exp[(r-0.5\sigma^2)(T-t)+\sigma(-\varepsilon)\sqrt{T-t}]$$
$$=100\exp[(0.1-0.5\times0.25^2)\times1+0.25(-\varepsilon)\sqrt{1}]$$
$$\tilde{C}=\max(0,S_T^{i'}-X)$$

（3）计算 $\hat{C}$ 和 $\tilde{C}$ 的平均值

$$C=\frac{1}{2}(\tilde{C}+\hat{C})=\frac{1}{N}\sum_i\left(\frac{\hat{C}_i+\tilde{C}_i}{2}\right)$$

（4）模拟 n 次并求 $\hat{C}=\max(0,S_T^i-X)$ 的平均值和 $\tilde{C}=\max(0,S_T^{i'}-X)$ 的平均值，两者的算术平均值贴现后即为所求的期权价格。

Python 函数调用如下：

```
from NumPy import *
res = doqqdj(100,100,1.0,0.1,0.25,5000)
print "res = ",res
```

得到如下结果：

```
res =  14.8926713193
```

根据 $C=SN(d_1)-X\exp(-r_fT)N(d_2)$，编制 B-S 期权定价公式 Python 函数如下：

```
def bscall_option(S,X,rf,sigma,T):
  d1 = (log(S/X) + (rf + 0.5 * sigma * * 2) * T)/(sigma * sqrt(T))
  d2 = d1 - sigma * sqrt(T)
  C = S * norm.cdf(d1) - X * exp( - rf * T) * norm.cdf(d2)
  return C
```

调用此函数

```
from SciPy.stats import norm
S = 100.0;X = 100.0;rf = 0.1;sigma = 0.25;T = 1.0
res = bscall_option(S,X,rf,sigma,T)
print "res = ",res
```

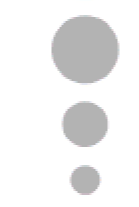
结果如下：

```
res =  14.9757907783
```

将这里的 B-S 期权定价 Python 函数计算结果 14.9757907783 与上述的对偶技术模拟结果 14.8926713193 进行比较可见基本上接近。

13.7.2 控制变量法

控制变量法就是将与所估计的未知变量密切相关的另一个已知量的真实值和估计值之间的差异作为控制量，以提高估计精度。在定价实践中，将两种衍生证券用相同的随机抽样样本和时间间隔，实施同样的蒙特卡罗模拟过程，能够得到两个模拟估计值，以第二种衍生



证券真实值与估计值之间的差异作为控制变量，最后得到第一种衍生证券的蒙特卡罗估计值。

假设 V_1 是需要估计的第一种衍生证券的价值，V_2 是价值容易估计的第二种衍生证券的价值，第一种证券与第二种证券相似，而 $\hat{V}_1$ 与 $\hat{V}_2$ 分别是第一种衍生证券和第二种衍生证券在同样的随机抽样样本的蒙特卡罗估计值，那么利用控制变量技术得到第一种衍生证券的价格估计值为

$$\hat{V}_1^{C1} = \hat{V}_1 + (V_2 - \hat{V}_2)$$

这里，$V_2 - \hat{V}_2$ 就是控制变量，它实际上是第二种衍生证券的蒙特卡罗模拟的估计误差，且上述方程的方差之间的关系为

$$\mathrm{var}(\hat{V}_1^{C1}) = \mathrm{var}(\hat{V}_1) + \mathrm{var}(\hat{V}_2) + 2\mathrm{cov}(\hat{V}_1, \hat{V}_2)$$

如果 $\mathrm{var}(\hat{V}_2) < 2\mathrm{cov}(\hat{V}_1, \hat{V}_2)$，一定有

$$\mathrm{var}(\hat{V}_1^{C1}) < \mathrm{var}(\hat{V}_1)$$

因此，当两种衍生证券的协方差很大时，或者当两种衍生证券的价格高度相关时，上述关系是成立的，两种衍生证券的正相关性越强，估计效率就越理想。然而从实际应用的角度看，这种控制变量技术的应用十分有限，因此，下面是更一般的控制变量技术，其控制变量的形式为

$$\hat{V}_1^{\beta} = \hat{V}_1 + \beta(V_2 - \hat{V}_2)$$

方差为

$$\mathrm{var}(\hat{V}_1^{\beta}) = \mathrm{var}(\hat{V}_1) + \beta^2 \mathrm{var}(\hat{V}_2) - 2\beta\mathrm{cov}(\hat{V}_1, \hat{V}_2)$$

这是关于控制变量系数 β 的二次三项式，下面的目标是能够找到特殊的 β 使方差 $\mathrm{var}(\hat{V}_1^{\beta})$ 最小，这时只要取 $\beta = \dfrac{\mathrm{cov}(\hat{V}_1, \hat{V}_2)}{\mathrm{var}(\hat{V}_2)}$，就可以保证方差 $\mathrm{var}(\hat{V}_1^{\beta})$ 最小。这种控制变量技术的缺点是 β^* 需要提前知道协方差 $\mathrm{cov}(\hat{V}_1, \hat{V}_2)$ 的信息，而这一般需要靠经验实现。

我们编制控制变量法的 Python 函数如下。

```
def bscall_option(S,X,r,sigma,T):
 d1 = (log(S/X) + (r + 0.5 * sigma * * 2) * T)/(sigma * sqrt(T))
 d2 = d1 - sigma * sqrt(T)
 C = S * norm.cdf(d1) - X * exp( - r * T) * norm.cdf(d2)
 return C

def kzqqdj(S1,S2,X,T,r,sigma,N):
 BSC = bscall_option(S2,X,r,sigma,T)
 z1 = random.standard_normal(N)
 z2 = random.standard_normal(N)
 ST1 = S1 * exp((r - 0.5 * sigma * sigma) * T + sigma * z1 * sqrt(T))
 ST2 = S2 * exp((r - 0.5 * sigma * sigma) * T + sigma * z2 * sqrt(T))
 hT1 = maximum(ST1 - X,0)
 hT2 = maximum(ST2 - X,0)
 C1 = exp( - r * T) * sum(hT1)/N
```

```
C2 = exp( - r * T) * sum(hT2)/N
return C1 + BSC - C2
```

例 13-6 考虑不支付红利股票的欧式看涨期权，它们的标的资产价格是 100 元，执行价格是 100 元，无风险利率是 10%，年波动率是 25%，期权的有效期是 1 年，用对偶变量法计算其欧式看涨价格。

解：在本例中，$S=100, X=100, r=0.1, \sigma=0.25, T-t=1$。

利用控制变量法计算期权价格的步骤如下。

（1）由 B-S 期权定价公式给出欧式看涨期权的价格 V_2。

（2）由蒙特卡洛法计算$\hat{V}_1$ 与$\hat{V}_2$

$$S_T^i = S_t\exp[(r-0.5\sigma^2)(T-t)+\sigma\varepsilon\sqrt{T-t}]$$
$$= 100\exp[(0.1-0.5\times0.25^2)\times1+0.25\varepsilon\sqrt{1}]$$
$$S_T^{i'} = S_t\exp[(r-0.5\sigma^2)(T-t)+\sigma\varepsilon\sqrt{T-t}]$$
$$= 100\exp[(0.1-0.5\times0.25^2)\times1+0.25\varepsilon\sqrt{1}]$$

从正态分布中随机抽取一个样本 ε 就给出一个 S_T^i 和一个 $S_T^{i'}$ 的值，取 $\max(0, S_T^i-X)$ 和 $\max(0, S_T^{i'}-X)$，则每模拟一次就得到两个欧式看涨期权在 T 时刻的值。假设模拟次数 5000，则分别给出 5000 个 S_T^i 和 5000 个 $S_T^{i'}$ 的值，从而有 5000 个 $\max(0, S_T^i-X)$ 和 $\max(0, S_T^{i'}-X)$，分别计算它们的算术平均值并无风险利率贴现，就得到了欧式看涨期权的两个估计值$\hat{V}_1$ 与$\hat{V}_2$。

（3）将上述所得结果代入$\hat{V}_1^{C1}=\hat{V}_1+(V_2-\hat{V}_2)$，得出欧式看涨期权的估值。

（4）模拟 n 次并求 $\max(0, S_T^i-X)$的平均值和 $\max(0, S_T^{i'}-X)$的平均值，两者的算术平均值贴现后即为所求的期权价格。

Python 函数调用如下：

```
from NumPy import *
from SciPy.stats import norm
S1 = 100;S2 = 100; X = 100.0;r = 0.1;sigma = 0.25;T = 1.0;N = 5000
res = kzqqdj(S1,S2,X,T,r,sigma,N)
print "res = ",res
```

得到如下结果：

```
res =  14.8921067991
```

练 习 题

对本章中的例题，在网上选择数据，使用 Python 中的工具重新操作一遍。

辅助资源说明

▶▶ 数据下载

尊敬的读者：

您好！感谢您选用清华大学出版社的图书！为帮助读者更加方便地学习本书，我们将书中实例的数据文件和其他数据资源打包收录，请您扫描二维码获取。



扫描二维码
获取数据资源

▶▶ 样书申请

为方便教师选用教材，我们为您提供免费赠送样书服务。授课教师扫描下方二维码即可获取清华大学出版社教材电子书目。在线填写个人信息，经审核认证后即可获取所选教材。我们会第一时间为您寄送样书。



任课教师扫描二维码
可获取教材电子书目



清华大学出版社

E-mail: tupfuwu@163.com　　网址：http://www.tup.com.cn/
电话：8610-62770175-4506/4340　　传真：8610-62775511
地址：北京市海淀区双清路学研大厦B座509室　　邮编：100084